

AD-A034 045

HUMAN RESOURCES RESEARCH ORGANIZATION ALEXANDRIA VA
BASIC COMPUTER PROGRAMMING: A SELF-INSTRUCTIONAL COURSE. (U)
JUN 67 R J SEIDEL, H G HUNTER, I C ROTBERG

F/G 9/2

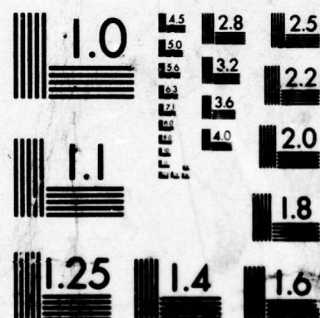
DA-44-188-ARO-2

UNCLASSIFIED

NL

1 OF 2
AD
A034045





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ADA034045

Approved for public release;
distribution unlimited

BASIC COMPUTER PROGRAMMING:

A Self-Instructional Course

R.J. Seidel
H.G. Hunter
I.C. Rotberg
W.A. Carpenter

June 1967

HumRRO Division No. 1
(System Operations)

The George Washington University
HUMAN RESOURCES RESEARCH OFFICE
300 North Washington Street
Alexandria, Virginia 22314

Approved for public release;
distribution unlimited

Work Unit METHOD
Sub-Unit II

ACCESSION for	
RTIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION.....	
BY.....	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	AVAIL. CODES SPECIAL
A	

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Basic Computer Programming: A Self-Instructional Course, Answer Book		5. TYPE OF REPORT & PERIOD COVERED (9) Instruction Manual,
7. AUTHOR(s) (10) R. J. Seidel, H. G. Hunter, I. C. Rotberg and W. A. Carpenter		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Human Resources Research Organization (HumRRO) 300 North Washington Street Alexandria, Virginia 22314		8. CONTRACT OR GRANT NUMBER(s) (15) DA-44-188-ARG-2
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Army, Office of the Chief of Research and Development Washington, D.C.		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE 4 2J024701A712
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 172p.		12. REPORT DATE (12) Jun 1967
		13. NUMBER OF PAGES 137
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES This research was performed by HumRRO Division No. 1 (System Operations) (now Eastern Division) under Work Unit METHOD II.		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer Programming Self-Instructional		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The aim of the course is to provide understanding of fundamental computer programming concepts, and more important, to develop a proficiency in writing elementary computer programs. The material was originally drawn from the U.S. Army's course for the Automatic Data Processing Specialist (ADPS) Programing Specialist (MOS 745.1). The criterion problems contained in the course were chosen from actual job situations, but scaled down to fit the instructional repertoire of the course. The course is self-contained and consists of an (continued...)		

DD FORM 1473

1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)


405 260

mt

20. Continued....

instruction booklet and an answer booklet. It is directed toward the level of high school seniors and first-year college students. In its various experimental forms it was administered to a total of nearly 900 high school juniors and seniors, and after final revision, to a small sample of college freshmen and high school seniors.

Findings from the several experimental administrations of the course clearly pointed to the desirability of teaching the course with a variety of kinds of computer problems. Results indicated that students learned from making errors and thus required hints or correct answers in solving the practice problems only a small part of the time.



Unclassified

FOREWORD

The self-instructional course in Basic Computer programming was prepared by the Human Resources Research Office under Work Unit METHOD, Research for Programed Instruction in Military Training. The course was developed as a vehicle for pursuing the objectives of Sub-Unit METHOD II--developing guidelines for applying programed instruction to military tasks requiring the learning of principles and rules. The task of programming for automatic data processing systems was selected as the research vehicle both because it typifies the use of principles and rules and because of the increasing demand for the skills of training computer programmers.

The aim of the course designed by HumRRO is to provide understanding of fundamental computer programming concepts, and more important, to develop a proficiency in writing elementary computer programs. The material was originally drawn from the U.S. Army's course for the Automatic Data Processing Specialist (ADPS) Programing Specialist (MOS 745.1). The criterion problems contained in the course were chosen from actual job situations, but scaled down to fit the instructional repertoire of the course.

The course is self-contained and consists of an instruction booklet and an answer booklet, separated from the main text for pedagogical reasons. It is directed toward the level of high school seniors and first-year college students. In its various experimental forms it was administered to a total of nearly 900 high school juniors and seniors, and after final revision, to a small sample of college freshmen and high school seniors.

The rationale for the approach used in developing the experimental course and the results of its administration are given in "The Application of Theoretical Factors in Teaching Problem Solving by Programed Instruction," by Robert J. Seidel and Harold G. Hunter, HumRRO Technical Report in preparation.

The research for METHOD II was conducted at HumRRO Division No. 1 (System Operations); course development took place during the period from 1962 to 1965. Dr. Arthur J. Hoehn was Director of Research when METHOD II was initiated; Dr. J. Daniel Lyons is the present Director of Research. Dr. Robert J. Seidel was the Work Unit Leader. HumRRO staff members who contributed to the research include Dr. Iris C. Rotberg, in the early stages; Dr. Eugene F. MacCaslin, task analysis; Dr. Donald Reynolds, Dr. Harold Wagner, and Dr. Richard D. Behringer, data collection; Dr. Harold G. Hunter, course preparation and report preparation. SP 4 Wayne A. Carpenter assisted in course preparation.

The superintendents of the school systems in the metropolitan Washington, D.C. area, who provided facilities and students for the administration of the experimental programed instructional course,

were: Dr. J.C. Albohm, Alexandria, Virginia; Dr. H. Wilson, Associate Superintendent for Instruction, Arlington County, Virginia; Dr. C.F. Hansen, Washington, D.C.; Mr. E.C. Funderburk, Fairfax County, Virginia; Dr. H. Elseroad, Montgomery County, Maryland; and Mr. W.S. Schmidt, Prince Georges County, Maryland.

HumRRO research for the Department of the Army is conducted under Contract DA 44-188-ARO-2 and Army Project 2J024701A712 01, Training, Motivation, Leadership Research.

Meredith P. Crawford
Director
Human Resources Research Office

CONTENTS

	Page
Introduction.	1
Phase I: Basic Operations.	1
Part One: Names and Places	1
Part Two: Moving Numbers	4
Part Three: Addition and Subtraction	10
Part Four: Address Arithmetic.	16
Part Five: Review and Practice	21
Phase II: Basic Looping.	25
Part One: Completing the Loop.	26
Part Two: Counting the Loops	32
Part Three: Getting Out of the Loop.	36
Part Four: Looping for Simple Addition	42
Part Five: Program Preparation	50
Part Six: Review and Practice.	56
Phase III: Data Processing	63
Part One: Address Modification	63
Part Two: Sort-and-Count Problems.	71
Part Three: Sorting Techniques	80
Part Four: Multiple Address Modification	89
Part Five: Review and Practice	96
Phase IV: Advanced Techniques.	101
Part One: Transfer Techniques.	101
Part Two: Index Registers.	107
Part Three: The LOD Command.	116
Part Four: The TRX Command	120
Part Five: The LDX Command	127
Part Six: Review and Practice.	132

PREFACE TO THE INSTRUCTOR

The course is directed toward the level of high school seniors and first-year college students. Its subject matter was originally drawn from the U.S. Army's course for the Automatic Data Processing Specialist (ADPS) Programing Specialist (MOS 745.1); the requirement for that course is a high school education or its equivalent. The aim of the course is to provide understanding of fundamental computer programming concepts, but even more important, to develop a proficiency in writing elementary computer programs. The criterion problems were chosen from actual job situations, but scaled down to fit the instructional repertoire of the course.

The structure of the course is linear but non-Skinnerian (e.g., the frame concept is not applicable). In fact, one of the purposes in developing the course was to use it as a research vehicle in which the effectiveness of a number of learning variables could be investigated. Consequently, the clearest way to describe the organization of the course is in terms of the number of concepts covered, and these are listed on the Contents page. As it stands, the course is self-contained and consists of an introductory section and about 140 pages of programed instruction. There is, in addition, an Answer Booklet, separated from the main body of the text for pedagogical reasons.

Findings from the several experimental administrations of the course clearly pointed to the desirability of teaching the course with a variety of kinds of computer problems (e.g., inventory, personnel selection, and salary computations). In addition, results indicated that students learned from making errors and thus required hints or correct answers in solving the practice problems only a small part of the time. How much aid is needed by any given student is undoubtedly a function of the individual's needs and abilities, and this is a topic for further study. *In implementing the course, we strongly urge that the student be cautioned not to use the separate Answer Booklet for support as he progresses through the material unless he honestly feels that he must do so.*

Development

In one revision or another the course was administered to over 900 students and military personnel. Since facilities were not readily available at the appropriate Army training installation (U.S. Army Signal School, Fort Monmouth, N.J.), civilian high school seniors and juniors comprised the bulk of the developmental sample. Close agreement was obtained, however, between the selection requirement on the Army Classification Battery for the ADPS Programing Specialist course and the scores of the civilian sample. In its final revision the course was administered successfully (final criterion scores averaged in the high 80s, with none lower than 77%) to 13 college freshmen and

high school seniors. Median completion time for the college freshmen was 26 hours, with daily sessions self-paced and lasting from three to five hours; median course duration for the high school students, whose scores averaged slightly lower than the college students, was 31 hours. The general attitude of the students toward the course was favorable, and they felt they had learned something that would be useful to them.

Content

Before describing the course in detail, it should be emphasized again that the entire context is oriented towards the writing of computer programs from the very beginning of the course. That is, a concept is introduced and then a problem incorporating this concept is presented which the student must program. These problems were taken from actual computer programming problems encountered on the job. It is this problem-oriented facet of the course which makes it rather different from most of the other programmed booklets available.

Prior to the introduction of each of the concepts, the student is given a preview of things to expect. Then, as he is introduced to the concepts, he is given short completion questions with the answers in light type on pages opposite to the questions. The problems to be programmed appear in a box-like format, and the student is expected to answer the problem whenever possible without looking up the answer first. At the end of each section of the book the student is given a review and then led on to the next set of concepts.

Because it was desirable to make the course available for use by the Army, the repertoire of commands and the symbolic language in which the computer programs are written have been chosen from material previously used by the Army to teach students how to program the MOBIDIC computer. The symbolic language is called FIELDATA Symbolic Language, which, although it represents a set of symbols interpretable by the MOBIDIC, is sufficiently general that it largely applies to almost any other computer and to the English language (the addresses).

An overview of the concepts of computer programming is given in the Introduction. Specific attention is paid to the concepts of command, address, the accumulator, memory locations, input and output, brief description of how a computer functions, and some sample mnemonics as they are used in computer commands. The coverage of these topics is interspersed with periodic review and with questions to which the student provides covert answers. He is instructed to continue reviewing the material preceding each of the sets of questions until he can answer the material to his own satisfaction. Finally, a short sample program is worked through for the student.

The course proper is separated into four phases or levels. In the first phase, Basic Operations (five parts), the student is introduced to storage locations, the concept of a central work area, the movement of numbers from place to place, and simple arithmetic operations. Next, Basic Looping concepts are covered (six parts) including general transfer commands, counting the loops, leaving the loops, and

program preparation. The third level, Data Processing (five parts) deals with address modification and address arithmetic (effective addressing), sorting and counting data into various categories, and multiple address modification. Finally, Advanced Techniques for transferring data are given (six parts), including indexing. In addition to the many computer programs to be written following the completion of each phase or level, the student practices on problems within each section of the course. In other words, he "learns by doing" throughout the course.

INTRODUCTION

The course you are about to begin will provide you with enough background to enable you to solve problems through the use of computers. Contrary to what many people may think, computers cannot operate by themselves. They must have human control to tell them *exactly what* to do *and when* to do it. Once they have been instructed exactly, computers can automatically perform the necessary operations for problem solution. The process of preparing the exact step-by-step instructions so that the computer can carry out the operations for problem solution is called programming. The entire series of instructions needed for each problem solution is called a program. You, as the person who will prepare these series of instructions or programs, are called a programmer. Typical problems that a computer deals with are stock control problems and mathematical formulas.

During the course you will learn a set of commands that will enable you to instruct a computer. You will become familiar with a number of terms that are common to the field of computer programming. You will learn, for example, that every instruction written for a computer consists of two parts: a *command*, which tells the computer what to do; and an *address*, which has information or content and tells the computer where specific figures are located.

You will also become familiar with the terms such as accumulator, memory, input, and output. The *accumulator* is a central working area where numbers are processed. *Memory locations* are actual, small physical units in which are kept all the figures you must work with. The address portion of each instruction word you write for a computer directs the computer to the memory location where specific figures are kept or are to be kept. *Input* refers to the means by which you can place your program into the computer. In this course, input refers only to punched cards; actually, a punched card is only one of the many types of input that computers can use.

Output is the record of the problem solution that the computer has accomplished. You will always get the output in some form of a written copy of the actual processing and the final values of the problem solution.

It may be helpful for you to think of your job as a programmer in terms of a post office employee who has to process mail. You sit at a central desk (the accumulator area); you process the mail by sorting it and weighing it so see if the postage is correct (prepare instructions on a program for computation and placement of information); each bit of mail has its own *address*, therefore, you must place it in the appropriate *memory* location. If the mail consists solely of packages, then you process it one way; if it is all first class, you process it another way; and so on. If the mail is mixed, you must consider different processing for each kind. In the same way, each specific problem you work with as a programming specialist will involve a particular mixing of kinds of mail. Consequently, *each* problem will require a *particular type* of processing or *program* based upon the complexities of the problem.

Now, on to a General Description of Computer Programming.

General Description

Most of us associate the word "computer" with "electronic brain" because the latter term is most widely used to describe what a computer is and what it does. However, the term "electronic brain" often creates misconceptions. Some people believe a computer can perform certain acts that it really cannot. One misconception is that computers can think and, therefore, are capable of making decisions. Actually, a computer *cannot* perform by itself. Like any other machine, it must be controlled. Of course, a computer can perform operations automatically--such as adding, subtracting, multiplying, and dividing--but it will only perform these functions when you operate the proper controls.

Computer controls are different from controls of other type machines. For example, to add a column of numbers with an ordinary adding machine, you have to press certain keys and operate certain levers to insert each number into the machine. To add the same column of numbers with a computer, all you do is insert a punched card (a card containing holes) into the machine. Then you press a button and the total appears immediately at the output. (You will learn later on that input information can be in several forms, but assume for now that cards are the only type of input.)

Review

What you have learned so far is that (1) a computer cannot perform by itself--it must be told what to do; (2) a computer can perform many operations automatically--it can add, subtract, etc.; (3) you must instruct the computer to perform these operations by inserting prepunched cards into the machine; and (4) the computer carries out the instructions almost instantaneously according to your controlled input and the results appear at the output of the machine.

Of course, this explanation is simplified. In actual practice, operating the computer is somewhat more involved. The instructions you write for the computer must be carefully planned. In fact, you have to plan these instructions in much the same manner as you plan a series of instructions that tell someone step-by-step how to perform a certain task. Therefore, we will first go over the basic procedures of how to write instructions for a person. Then we will use these basic procedures to learn how to write instructions for computers.

The instructions you write for a person to perform a certain task must be written in easy-to-understand steps and the steps must be in the same sequence in which they are to be performed. For example, suppose you write a series of instructions explaining how to calculate the hourly pay rate for each person in your company. There are two ways to write these instructions. One way is to write them in detail as shown below.

1. Determine (abbreviated GET) the number of hours worked in a day.
2. Multiply (abbreviated MLY) number of hours per day by number of days worked per month.
3. ADD number of weekend KP hours worked.
4. Subtract (SUB) number of pass hours received in a month.
5. Record, or store (STR), the total hours in some convenient place.
6. GET base pay per month.
7. ADD allowances.
8. Subtract (SUB) income tax, social security, and so on.
9. Divide (DVD) total pay by total hours.

The other way to write instructions is to use key words and abbreviations as follows:

<u>Command</u>	<u>Information</u>
1. GET	DAY HOURS
2. MLY	MONTH DAYS
3. ADD	KP HOURS
4. SUB	PASS HOURS
5. STR	TOTAL HOURS
6. GET	BASE PAY
7. ADD	ALLOWANCES
8. SUB	DEDUCTIONS
9. DVD	TOTAL HOURS

QUESTIONS

1. When you write instructions to have someone perform a task, how should you arrange these instructions?
2. Here are three instructions written in detail. How would these same instructions appear when key words and abbreviations are used?
 - (1) ADD number of weekend KP hours worked.
 - (2) Subtract (SUB) number of pass hours received in a month.
 - (3) Record, or store (STR), the total hours in some convenient place.

Note that by using key words, the abbreviated instructions give the same information as the detailed instructions. Notice too that by using key words, each instruction is divided into two parts. One part gives a *command*, the other part gives *information*. However, to solve the problem you need actual figures. That is, to calculate a person's hourly pay rate, you must know the number of hours he works in a day, the number of days he works in a month, and so on. You have to consult some record or go to some location to get the specific figures that pertain to each person's salary and work. For example, suppose Ed Baker is a supply clerk. To find out how many hours he works in a day and how many days he works in a month, you have to consult the supply sergeant's duty roster. To find out how many hours of KP he works, you have to consult the KP roster. To learn how many pass hours he gets in a month, you have to consult the first sergeant's leave roster. In other words, the key words in the *information* part of each instruction represent a *location* where specific information is stored. Therefore, the information part of each instruction is actually called the ADDRESS. This two part instruction is similar for computer instructions, as explained in the next paragraph.

QUESTIONS

1. Do key words give less, the same, or more information than detailed instructions?
2. How many parts does an instruction have basically? What are they?
3. Where must you obtain the specific information needed in the various steps of your instructions?

Computer instructions, like the abbreviated human instructions, are divided into two parts: the *command* part and the *address* part. Each command, as in the human instructions on Page 5, is a three-letter word or abbreviation; and each address specifies a location where information is stored. However, computer instructions differ slightly from human instructions. You will learn how they differ by comparing the two parts of each type instruction as described next.

Computer commands are very like the abbreviated human commands given on Page 5 except for the command GET. In place of GET, the MOBIDIC computer (the machines you will work with later on) uses CLA. These letters are an abbreviation for the command *CLear and Add*, which tells the computer to clear from its work area (called the accumulator) the results of the previous computation and copy into it the information from the location specified by the address. The MOBIDIC also uses several other commands that differ from the commands you may give a person; you will learn these commands in detail later.

QUESTIONS

1. How are computer instructions similar to human instructions?
2. What command does the MOBIDIC computer use instead of GET?
3. How many letters are in a computer command?

Review

Earlier we noted that a computer can perform many operations automatically if it is given the proper set of instructions, called a program, by insertion of prepunched cards into the machine. The material we have just covered defined a proper instruction as consisting of two parts: a *command* and an *address*. The three-letter *command* tells the machine what to do, and the *address* specifies a *memory location* where the information to be processed is stored. Recall the similarity to the post office. The *commands* enable you to process for first class, parcel post, and other kinds of mail while each piece of mail has its own *address* and must be placed in a specific memory location.

Now you will go on to learn how the address part of computer instructions and human instructions compare. The *address* part of a computer instruction, like the address part of a human instruction, specifies a location. Computer locations, however, differ from the locations in human instructions because a computer location is actually part of the computer. A human being could either remember all the information or he could write it on a scratch pad. The computer has a scratch pad or storage area called the MEMORY. This memory area is divided into many small units called LOCATIONS, which store information electrically.

The information you store in the individual memory locations specified by the *address* part of each instruction must, of course, be the exact figures you want the computer to work with. For example, assume that Ed Baker works eight hours a day, 22 days a month. On the average he gets a three-day pass each month (24 hours off) and he works 12 hours on KP each month. His base pay is \$200 a month and he gets \$50 in various types of allowances. He pays \$30 for combined income tax and social security. These figures must be stored in the locations specified by the *address* of each instruction, as shown in Table 1 and in Figure 1 on Page 15.

QUESTIONS

1. What does the ADDRESS part of a computer instruction specify?
2. What does computer MEMORY mean?
3. What character must the information noted in the ADDRESS have?
(Hint: General or specific?)

TABLE 1: Ed Baker's Pay Record

LOCATION	SPECIFIC VALUE STORED
HOURS	8
DAYS	22
KP	12
PASS	24
BASPAY	200
ALLOW	50
TAX	30

FIGURE 1: Ed Baker's Pay Record as it might be represented in a computer's MEMORY

HOURS (8)	
	BASPAY (200)
TAX (30)	
	PASS (24)

	TOLHRS (?)
DAYS (22)	
KP (12)	
	ALLOW (50)

With these figures stored in the proper memory location, the computer can perform the calculations step-by-step, as described in the instructions. Exactly how the computer executes each instruction is explained next. Note first that the instructions must also be stored in memory as shown in Figure 2 below. The instructions are always stored in sequence in memory.

The Information As It Is Stored In Memory		Instructions As They Are Stored In Memory	
		TOLHRS (?)	CLA HOURS DVD TOLHRS
HOURS (8)			MLY DAYS
	BASPAY (200)	DAYS (22)	ADD KP
		KP (12)	SUB PASS
TAX (30)		ALLOW (50)	STR TOLHRS
	PASS (24)		CLA BASPAY
			ADD ALLOW
			SUB TAX

FIGURE 2

To execute each instruction a computer must perform each step or operation in much the same manner as a person does. For example, to execute the instructions below, you need a scratch pad and pencil to keep track of the sequence of calculations and numbers you use in the calculations. You also need the scratch pad to record the results.

COMMAND

1. GET (for person)
2. MLY
3. ADD
4. SUB
5. STR
6. GET (for person)
7. ADD
8. SUB
9. DVD

INFORMATION

DAY HOURS
MONTH DAYS
KP HOURS
PASS HOURS
TOTAL HOURS
BASE PAY
ALLOWANCES
DEDUCTIONS
TOTAL HOURS

QUESTIONS

1. How are instructions stored in memory?
2. Is the execution of computer instructions similar or dissimilar to that of human instructions? In what way is it similar or dissimilar?

A computer must keep track of the numbers it uses in a calculation, and it must hold the results of that calculation until it receives an instruction as to what it should do next. In other words, the computer also must have some sort of scratch pad. The particular device that a computer actually uses to hold the results of calculation is called the *accumulator*, which is simply a *temporary storage device* similar to an individual memory location. The number of temporary storage devices that a particular computer uses depends on the size of the computer. Some small computers may use only one, whereas large computers may use seven or more. To get a more complete picture of what the accumulator does, follow the step-by-step description below which explains how a computer with one temporary storage device executes the instructions shown here.

<u>COMMAND</u>	<u>ADDRESS</u>
1. CLA (for computer)	HOURS
2. MLY	DAYS
3. ADD	KP
4. SUB	PASS
5. STR	TOLHRS
6. CLA (for computer)	BASPAY
7. ADD	ALLOW
8. SUB	TAX
9. DVD	TOLHRS

QUESTIONS

1. Does a computer keep track of its calculations? How?

1. CLA HOURS. This instruction tells the computer to clear the accumulator; then to location HOURS and put the number it finds there into the accumulator. Table 1 shows that the number 8 is stored in location HOURS, so the accumulator now contains 8.

2. MLY DAYS. This instruction tells the computer to go to location DAYS and multiply the number it finds in this location by the number in the accumulator. Location DAYS contains 22 and the accumulator contains 8. So the result of this instruction puts 22 times 8, or 176, into the accumulator.

3. ADD KP. Add the number you find in location KP to the number now in the accumulator. So 176 plus 12 from location KP equals 188. The number 188 now appears in the accumulator.

4. SUB PASS. Subtract the number in location PASS from the number in the accumulator; 188 minus 24 equals 164, which is now in the accumulator.

5. STR TOLHRS. Store the number now in the accumulator into location TOLHRS. So the number 164 goes into location TOLHRS.

6. CLA BASPAY. Clear the accumulator and add the number you find in location BASPAY. Location BASPAY contains 200; so the accumulator now contains 200.

7. ADD ALLOW. Add the number from location ALLOW to the number in the accumulator. The number 200 now in the accumulator plus 50 from ALLOW equals 250, the new number appearing in the accumulator.

8. SUB TAX. Subtract the number in location TAX from the number in the accumulator. So 250 minus 30 equals 220, the number now in the accumulator.

9. DVD TOLHRS. Divide the number in the accumulator by the number stored in location TOLHRS. Thus, 200 divided by 164 equals 1.35, Ed Baker's salary rate per hour, which is now in the accumulator.

QUESTIONS

1. What does the accumulator do?

Notice how these instructions tell the computer *what* to do and *when* to do it. The entire list of instructions, as noted earlier, is like a program of events, explaining *what* is to take place and *when*. For this reason, a list of computer instructions is actually called a program. It follows, then, that the process of writing these instructions is called *programming* and the person who writes the programs is called a *programmer*.

QUESTIONS

1. Why is the list of instructions given to the computer called a program?
2. What is the title of the person who writes the instructions?

Review

Now to make sure that you do not forget how you program the actions of a computer, review the important points, as described below.

Computers are machines; therefore, to make them do work you have to control their actions. You learned that you control computer actions with punched cards. In reality, using punched cards is only one way to control the operations of a computer. There are several other ways, but you will learn these later.

You arrange the punches on the cards by first writing out a series of instructions called a program. Each instruction tells the computer what to do in order to accomplish the objective of the program. The instructions must be arranged in the sequence that the computer is to follow. After you write a program, you type each instruction in sequence on a machine that works like a typewriter. Instead of typing letters on a page, this machine punches holes in cards.

The most practical type of problem for computers to solve is the type that must be solved, or processed, periodically. Examples are: payrolls, stock records, and mathematical formulas in engineering. The reason these problems are practical for computer processing is that the same calculations and the same sorting process must be repeated many times. The hourly pay rate problem is a good example because you have to perform the same calculation for each man. If your company were very large you would have to repeat the calculations several hundred times. By processing this problem on a computer you would go through these steps only once--at the time you write the program. After that, if you wish to solve this problem, all you do is insert the program into the computer. During this course you will learn how to write sets of instructions which will enable you to solve practical problems on a computer.

INSTRUCTIONAL NOTE

If you have completed the introductory material to your satisfaction you are now ready to begin the course.

Remember that this course consists of three different kinds of materials. You will be given text to read. Secondly, there will be completion problems, with answers in light type on the pages opposite to the incomplete sentences. Thirdly, there will be larger problems given to you in box format, and the answers to these problems are in the accompanying Answer Booklet. Read the material carefully and try to answer all problems as best you can before you look up the answer. Do not look up the answer unless you feel it absolutely necessary to do so.

PHASE I

BASIC OPERATIONS

Your instruction begins with the basic computer operations. These include the movement of numbers from one place to another, and their combination through addition and subtraction.

The section is divided into five parts:

Part One: Names and Places. This part talks about the places where numbers are stored in the computer, where they are acted upon, and the names for these places.

Part Two: Moving Numbers. Here you will learn how the computer can be told to move numbers from one place in the computer to another.

Part Three: Addition and Subtraction. At this point you will start working problems in simple arithmetic.

Part Four: Address Arithmetic. Part Four explains a convenient technique for naming a large number of storage places.

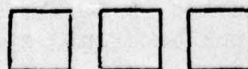
Part Five: Review and Practice.

PART ONE: NAMES AND PLACES



Numbers can be kept, or stored, at places in the computer called memory locations, or storage locations (they both mean the same thing):

Storage Locations



These memory locations can be given names. For example, they might be called TOM, DICK, and HARRY:

TOM	DICK	HARRY
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

When you start work as a programmer, you will be able to name storage locations as you choose. During your instruction, however, the names will be given for you.

Nevertheless, you should know the basic rules for assigning names, even though you won't use them until later. A storage location can be given any name that (1) has no more than six letters and numbers altogether, (2) starts with a letter, (3) has no spaces, and (4) has all letters capitalized.

SIXMEN is permissible, but SIX MEN is not. Why?

JOHN is permissible, but JOHNSON is not. Why?

THREE3 is permissible, but 3THREE is not. Why?

NUMBER is permissible, but Number is not. Why?

Which of the following are also not permissible: P, QXY, Benny, A1965, SYMBOLIC, 34567?

The name given to a storage location is called its address.

You can think of storage locations as little houses. Each house has its own address and contains a number. If you need a certain number, you go to its address, the place where it is contained.

Programmers also use "address" as a verb. To address a storage location means using its name in an instruction to the computer.

Numbers are "remembered" by the computer by storing them in memory locations, but they are combined and used in a different place, called the accumulator.

The accumulator is a central work area. Numbers are brought out of storage up to the accumulator, worked on there, and put back into storage.

storage,
memory

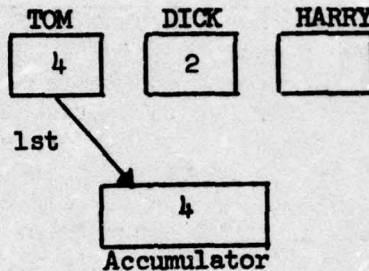
accumulator

address

Numbers are stored in the memory locations, but they are acted upon in the accumulator.

Let's say you wanted to add together the numbers stored in locations TOM and DICK. For this example we will assume that TOM contains a 4 and DICK contains a 2.

The first instruction to the computer would say, "Copy the number from TOM into the accumulator."



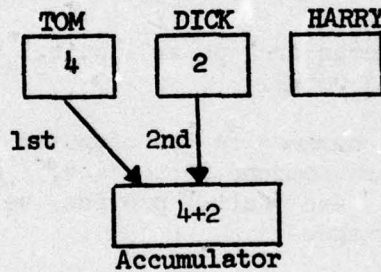
Space

Too many letters

The second instruction would say, "Add the number from DICK to the number already in the accumulator."

Starts with a number

Not capitalized



Benny,
SYMBOLIC,
34567

The sum of TOM+DICK would now be in the accumulator.

As the example shows, arithmetic operations like addition and subtraction take place in the accumulator using numbers brought up from the various storage locations.

To review, numbers are stored at places in the computer called ____ locations or ____ locations.

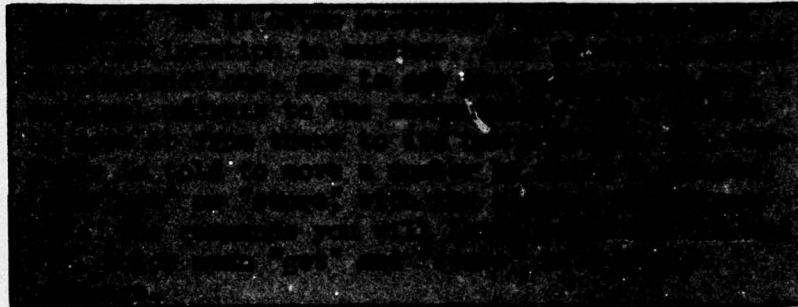
Locations can be given individual names. The name of a location is called its ____.

The central work area, where computation takes place, is called the ____.



clear

PART TWO: MOVING NUMBERS



accumulator

4

clear

add

You give commands whenever you say things like "Pass the salt," "Quit kidding," or "Throw the ball."

The action words are "pass," "quit," and "throw." They are command words.

CLA VALUE

3

We can give commands to the computer in much the same way. We can command it to "get," "add," "subtract," "store," and "halt," provided we talk to it in its own language.

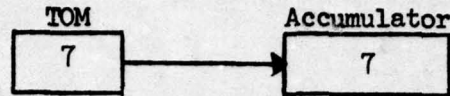
The word for "get" in computer language is CLA, which stands for "Clear and Add." You can guess, therefore, that the CL stands for _____ and the A stands for _____.

Next, we have to tell it what to get, that is, what number to "clear and add."

All commands (except the one for "halt") are used with addresses, so the computer will know what number to use with the command.

For example, CLA TOM tells the computer, "Get the number in TOM up to the accumulator." Assume TOM contains a 7.

CLA TOM



Recall that the CL in CLA stands for ____.

This means that the accumulator is cleared before the new number is copied in. Any previous number is erased or destroyed and is replaced by the new number.

Say the accumulator contains the number 6500, and the location with the address COST has a 4. What is in the accumulator after CLA COST?

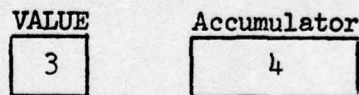
CLA COST says to clear the ____ and then copy in the number from COST.

CLA PRICE would tell the computer to ____ the accumulator of any number already there and then ____ in the number from PRICE.

If you wanted to get the number from VALUE up to the accumulator, you would write _____. Say the accumulator had a 4 and location VALUE had a 3. What would be in the accumulator after CLA VALUE-- 3, 4, or 7?

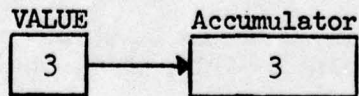
The 4 in the accumulator is erased, or cleared, and the 3 from VALUE is copied in, as illustrated below:

Before CLA VALUE



clear,
add

After CLA VALUE



(The 4 is erased first.)

Say the number from SAM was already in the accumulator when you told the computer to CLA BOB. What would then be in the accumulator--SAM, BOB, or SAM+BOB?

CLA erases the accumulator clear and then copies in the new number.

CLA SUITS

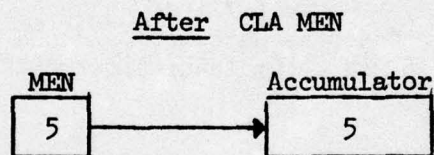
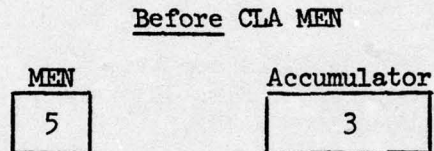
There is another feature of the CLA command that you should know. It only copies the number from the storage location, without changing it. CLA could be read, "Clear and copy."

no

Say there is a 5 in MEN and a 3 in the accumulator. What is in MEN after CLA MEN? What is in the accumulator?

replace

The last example might be diagrammed this way:



store

(The 3 is erased.)

After a CLA command, the number in the accumulator and the number in the location addressed will be the same.

Questions below are based on the following instructions:

COST

CLA THIS
CLA THAT

After these instructions have been performed, what is in the accumulator--THIS, THAT, or THIS+THAT?

PRICE

What is in location THAT at the completion of the two instructions--THIS, THAT, or THIS+THAT?

accumulator

What is in THIS--the number that was there at the beginning or a different number?

The CLA command clears out the accumulator and copies in the number from the location addressed; CLA clears and copies. For example, CLA COST clears the accumulator and copies in the number from COST.

BOB

What instruction will copy the number from SUITS into the accumulator?

Does the number from SUITS add to or replace the number already in the accumulator?

5

Is the number in SUITS changed?

Incidentally, an instruction to the computer is called just that—an instruction. CLA SUITS is an instruction. Each instruction has a command, such as CLA, and an address, like SUITS.

5

A series of instructions is called a program.

So far we have been moving numbers in one direction only, from storage locations into the accumulator. To move a number in the other direction, from the accumulator to a storage location, requires a new command.

The new command is STR. What do you guess STR stands for?

The STR command works in a way that is exactly the reverse of the CLA command.

With that information, see if you can answer the following questions by yourself. You will be contrasting what happens in the accumulator with what happens in the storage location addressed.

If CLA COST first clears the accumulator, STR COST first clears location ____.

THAT

If CLA PRICE copies the number from PRICE into the accumulator, STR PRICE copies the number from the accumulator into ____.

the same
number

Both CLA and STR only copy numbers, without changing them. If CLA VALUE copies the number from VALUE, STR VALUE copies the number from the ____.

THAT

STR

Think of it this way: CLA and STR are both clear-and-copy commands, but they differ in what they clear and copy, the accumulator or a storage location.

CLA clears the accumulator, while STR clears a

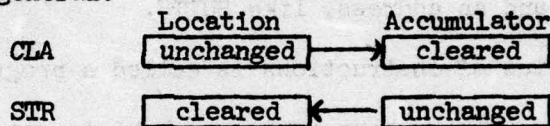
CLA

CLA copies the number from a storage location, while STR copies the number from the _____.

The _____ command can change a number in the accumulator, while the _____ command can change the number in a storage location.

The _____ command does not disturb the number in the accumulator, while the _____ command does not disturb the number in the storage location.

In general:



CLA COST
STR VALUE

For example:

	COST	Accumulator
to begin with	1	100
CLA COST	1	1
STR COST	100	100

Review: The CLA command clears the accumulator and copies in the number from the storage location addressed, while the STR command works in reverse fashion, clearing the storage location addressed and copying in the number from the accumulator.

Here are a few problems to test whether you understand how the CLA and STR commands work:

CLA HATS
STR GLOVES

There is a 6 in the accumulator and a 3 in TUBES. What instruction will change the number in TUBES? What instruction will change the number in the accumulator?

CLA SUITS
STR PANTS

CLA PAY1
STR PAY2
STR PAY3

You have a number in the accumulator you need to leave there for the next instruction. What command will save that number without disturbing it?

You have a number in the accumulator you would like to get rid of. What command will do it?

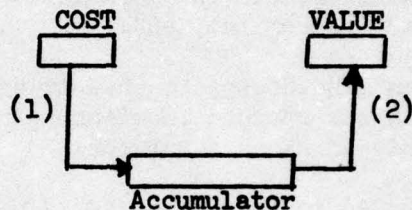
accumulator

Problems that require a number in one location to be copied into another location are called relocation problems, since they relocate numbers.

storage
location
(memory
location)

For example, say that the number in COST is to be "relocated" in VALUE. This requires two instructions, one to bring the COST number to the accumulator and a second instruction to store it in VALUE:

STR,
CLA



What two instructions will relocate COST into VALUE?

Here's the answer to the last question, with an explanation:

CLA COST	The accumulator is cleared and the number from COST is copied in; we now have COST in the accumulator.
STR VALUE	Location VALUE is cleared and the number from the accumulator, which is the number from COST, is copied in.

At this point, the original number from COST is in all three places—COST, VALUE, and the accumulator.

What two instructions will relocate SUITS into PANTS?

How would you relocate HATS into GLOVES?

CLA TUBES

Relocate the number from PAY1 into PAY2, and also into PAY3.

STR TUBES

PART THREE: ADDITION AND SUBTRACTION

CLA SHOES
ADD SHOES

Preview. The most difficult part of this section is now completed. If you understand thoroughly how the CLA and STR commands work, the rest of the commands will offer no problem. They are ADD, SUB, and HLT, which work in exactly the way you would expect.

accumulator

twice the
number

Incidentally, don't forget that all letters in an instruction are capitalized. It would be wise to get into the habit now.

The commands are all three-letter code words (with all letters capitalized).

The ADD command does just what it says. It adds the number in the storage location addressed to the number already in the accumulator.

If you wanted to add the numbers in COST and VALUE, the instructions would be:

1. CLA COST
2. — —

It makes no difference which number is brought to the accumulator first. You could just as easily turn the locations around and write (be careful of the command you use with the first instruction):

— —
— —

The ADD command does not disturb the number in the location addressed. ADD changes the number in the accumulator, but doesn't affect any of the memory locations.

What other command does not disturb any storage location?

If you wanted the accumulator to contain a number twice as big as the number in COUNT, what instructions would you write?

CLA CAR
ADD CAR
STR JAR

The ADD command doesn't change the memory location addressed. The number there can be added into the accumulator as many times as you like.

What instructions will multiply SHOES by 2, giving SHOES x 2? (Hint: $S \times 2 = S + S$)

Where will the answer be--in the accumulator or in SHOES?

Look at the following instructions carefully:

CLA COUNT
ADD COUNT
STR COUNT

What is in location COUNT at the end--the original number or twice that number?

Here's an explanation of the last problem:

CLA COUNT	The number from COUNT is copied into the accumulator; <u>the number is copied, not changed, and can be used again.</u>
ADD COUNT	Add the number from COUNT to the number in the accumulator; <u>a number equal to COUNT+COUNT is now in the accumulator.</u>
STR COUNT	The original number in COUNT is <u>erased</u> , and the number from the accumulator is put in; <u>the number in COUNT is now twice its original size.</u>

ADD VALUE

CLA VALUE
ADD COST

The sort of thing we just did is called a desk check. It means checking a program on paper to see that it works properly. Often this is done by assuming different numbers are contained in the various locations, and tracing out what the program does to those numbers.

Desk checks are quite useful. We will use them fairly often. You should get into the habit of running desk checks on your own.

What instructions will double the number in location CAR?

CLA

CLA COUNT
ADD COUNT

The STR command is the most difficult to understand of all the commands. It has two unique features. It is the only command that can change the contents of a memory location, and it is the only command that does not change the contents of the accumulator.

The first feature allows you to change the number at the same address many times. For example, what instructions will make PRICE equal to COST?

How could you then change PRICE to make it equal to MAN?

SUB DESK

The second feature, that STR does not affect the accumulator, allows you to store the same number in as many locations as you like. No matter how many times you store it, the number is still in the accumulator. Store THIS into THESE, THOSE, and THEM.

no

CLA, ADD

The ADD command adds the number in the location addressed to the number already in the accumulator. The number in the storage location is not changed. ADD MEN adds the number from MEN to the number in the accumulator without changing the number in MEN.

Now you will begin working actual programs. Write out the instructions for the following problems on a separate piece of paper. Answers appear in a separate booklet. Although you are permitted to turn to the answers at any time, you will do much better on the course if you do not use the answers unless absolutely necessary.

Problem 3.1. Write a program to add the numbers in EENY, MEENY, and MINY.

Problem 3.2. A battalion commander wants to know the number of men absent from duty last month. The number who were absent because they were on leave is in LEAVE, the number absent without leave is in AWOL, and the number on temporary duty elsewhere is in TDY. Write a program to compute the answer, putting it in location ABSENT.

STR

Problem 3.3. Double the number in VALUE.

A fourth command—in addition to CLA, ADD, and STR—is the SUB command. You can guess that it stands for ____.

The SUB command works as you would expect it to. It subtracts the number in the location addressed from the number already in the accumulator, leaving the answer in the accumulator. The number in the storage location is undisturbed.

CLA COST
STR PRICE

CLA MAN
STR PRICE

To subtract DESK from CHAIR, you would write:

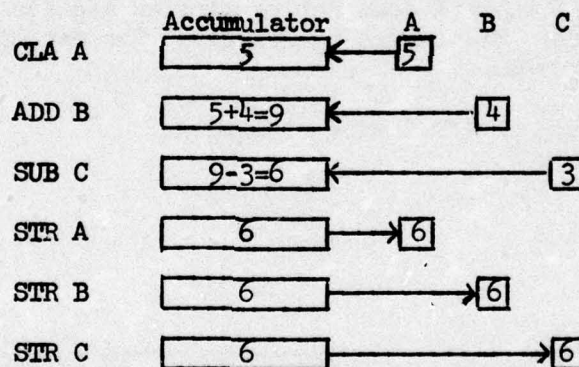
CLA CHAIR

Is the number in DESK different after the instruction SUB DESK?

CLA THIS
STR THESE
STR THOSE
STR THEM

The SUB command does not disturb the contents of the location addressed. For example, it does not disturb DESK in the instruction SUB DESK. What other two commands do not disturb the locations used with them?

Next, we will show you a program using all four commands, along with a schematic representation of what happens in the accumulator and in the various memory locations. Assume location A contains a 5, B a 4, and C a 3.



A command can change either the accumulator or a storage location, but not both. Which command can change the number in a storage location?

subtract

Numbers can be added or subtracted in the accumulator by the ADD and SUB commands. The answers appear in the accumulator. Numbers in the storage locations are not changed by these commands.

Write out the following problems on a separate piece of paper. Answers are in the separate booklet.

Problem 3.4. Find a man's net pay (NETPAY) by adding his regular pay (REGPAY) to his overtime pay (OVTIPAY) and subtracting out social security (SOCSEC) and income tax deductions (INCTAX).

Problem 3.5. Find the cost of a pen (PEN) if it equals the price of one pencil (PENCIL) minus two erasers (ERASER).

The last command you will learn for a while is HLT. It stands for "halt" and means just what it says.

To make the computer stop, simply write ____.

Notice how the command for "halt" is spelled. The vowel is dropped out, leaving the three letters

- - -

COST+MAN

The HLT command does not require an address to go with it. It is used by itself at the end of every program.

Complete the following program:

CLA TREE
ADD LEAF
STR WOOD
—

The HLT command is written at the end of every program. It is used by itself and stops the computer.

Work out the problems which follow, writing out the instructions on a separate piece of paper. Answers are in the separate booklet. Don't forget to HLT each program.

Problem 3.6. "Update" the stock level in a supply depot by adding the amount of stock received (RCVD) to the amount on hand (STOCK), subtracting the amount issued (ISSUE1, ISSUE2, and ISSUE3), and storing the result back in STOCK. Don't forget to stop the computer.

Problem 3.7. Compute and store both total pay and net pay using the following data:

Regular pay (REGPAY) + overtime pay (OVTPAY) = total pay (TOTPAY).

Total pay (TOTPAY) - deductions (DEDUCT) = net pay (NETPAY).

Students sometimes cling to the idea that the number at an address never changes. It can change and often does. See how PRICE changes below. First it changes to equal COST, then it becomes equal to COST+MAN.

1. CLA COST
2. STR PRICE (PRICE = COST)
3. ADD MAN
4. STR PRICE (PRICE = COST+MAN)

Incidentally, notice what is in the accumulator after each instruction in the last program:

HLT

H L T

1. Accumulator equals COST.
2. Accumulator still equals COST.
3. Accumulator = COST+MAN.
4. Accumulator = _____.



HLT

Write out the following problems separately. Answers are in the separate booklet.

Problem 3.8. "Updating" programs changes numbers to make them more current. For example, a personnel office determines daily strength by adding arrivals and subtracting departures. Update enlisted-women strength and enlisted-men strength, given the information below:

<u>Personnel</u>	<u>Current</u>	<u>Arrivals</u>	<u>Departures</u>
Enlisted women	EW	EWCOME	EWGO
Enlisted men	EM	EMCOME	EMGO

Problem 3.9. Relocate RECORD into location X, a number twice the size of RECORD into location Y, and a number three times the size of RECORD into location Z.

PART FOUR: ADDRESS ARITHMETIC

EXERCISE: Most computers use a system of addresses to store information. You number addresses, starting from 0. Each address is a letter and a number, like COST+5. This is a way of making up new addresses for each piece of information. Let's you store information in an address. It is called "address arithmetic."

PAY+99

Storage locations can be named by calling them things like COST, VALUE, and PRICE. These names are called "addresses," as you know.

Most problems you will encounter on the job will deal with a series of related pieces of information, each stored at a different location. For example, you may have to compute the total salary paid to all employees, where each employee's salary is stored in a different place.

You could make up a different address for each one of these locations, but if there are several hundred, it would be quite a job.

PAY+9
PAY+55
PAY+100

The easy way is to choose one basic name and add a different number for each location.

For example, if three locations are required for three different salaries, you could provide individual names like SALARY, PAY, and WAGE; or you could take the easy way and call them PAY, PAY+1, and PAY+2.

The latter method, using one basic name with different numbers, is called address arithmetic.

Recall that one of the rules for assigning names to memory locations was that the address could have no more than (how many?) letters and numbers combined.

The rule of six letters and numbers, or less, does not apply to address arithmetic.

Only the basic name, before the numbers are added on, must conform to the rule.

Since address names can include numbers, you could use names like PAY1, PAY2, and PAY3, which is different from PAY+1, PAY+2, and PAY+3.

Don't be confused by the word "arithmetic." It does not mean computation, like addition or subtraction. Address arithmetic is simply a convenient way to assign address names to memory locations.

Notice that the numbering starts with the second address in the series. If you were naming 100 locations, the first might be PAY, the second PAY+1, the third PAY+2, and so on. What would be the address for the last, or 100th, location? (Be careful; it is not PAY+100.)

The last address in an address arithmetic series has a number that is one less than the number of addresses in the series. The last address in a series of 100 addresses relative to PAY would be PAY+99 (99 addresses with numbers, plus the first one which has no number after it).

The first address is not PAY+1, but simply PAY. If you have only three addresses in the series—PAY, PAY+1, and PAY+2—the third is PAY+2. What would the 10th address be? The 56th address? The 101st?

Now that you are beginning to understand address arithmetic as a system for numbering a series of storage locations, we can begin using it in some problems. Write out the following programs on a separate piece of paper. Answers are in the separate booklet.

Problem 4.1. Write a program to compute and store net pay using these address locations:

Regular pay----RECORD
Overtime pay---RECORD+1
Deductions-----RECORD+2
Net pay-----RECORD+3

(Did you stop the computer?)

Problem 4.2. Write a program to relocate a company's records from one month to the next. The number of employees is in WORKER, their total salary is in SALARY, and their total tax deductions is in TAXES. Relocate these figures into the locations for the next month, which are WORKER+1, SALARY+1, and TAXES+1.

The use of address arithmetic tells the computer to reserve adjacent storage locations, or locations that are beside each other, forming a series in the computer.

Thus, if RECORD+3 were used in an instruction, the computer would go to, or "call on," the memory location three addresses removed from RECORD. Would RECORD+3 be the third or the fourth address in the series?

Incidentally, when you hear a programmer talk about "a series relative to COUNT," he means the series of addresses: COUNT, COUNT+1, COUNT+2, and so on, using COUNT as the basic name.

Here are a few more problems using address arithmetic. Again, the answers are in the separate booklet.

Problem 4.3. Information on an employee's history with a company is stored relative to MAN. The date he was hired is in MAN, his starting salary is in MAN+1, and his present salary is in MAN+2. Relocate the information into the series relative to WORKER.

Problem 4.4. The total stock in a supply depot is computed at the end of each day. The number of items received Monday is in location GET, for Tuesday it is in GET+1, and for Wednesday it's in GET+2. Similarly, the items issued on those days is in the series relative to ISSUE. Store the stock level at the end of each day in the series relative to STOCK, where STOCK contains the stock level at the start of Monday. (Hint: Start with STOCK, add GET, subtract ISSUE, and store the result in STOCK+1.)

The real advantage of address arithmetic is that the computer can be given special instructions which say, "Call on all locations in such-and-such an address arithmetic series."

In other words, you can assign address arithmetic names to an entire series of locations, and then use just the one basic name, with instructions, to visit all other locations in that series.

You can't do that when the addresses represent different names, like COST1 and COST2, but only when an address arithmetic series is formed by the "+" sign with a number, such as COST, COST+1, COST+2, and so on.

You will not learn this special technique until later in your instruction. For now, simply notice the difference between using numbers for naming individual locations (A42, MAN678, TAKE5, and the like) and using numbers with a + sign for naming locations in an address arithmetic series (such as PLACE, PLACE+1, PLACE+2, and so on).

Now try a few more problems. Write them out on a separate piece of paper.

Problem 4.5. A company keeps a three-word record on each employee, stored relative to EMPLOY. This means it keeps three types of information, stored in three consecutive locations in the address arithmetic series relative to EMPLOY. The "words" are: marital status, years with the company, and ID number:

	Marital status	EMPLOY
1st employee	Years service	EMPLOY+1
	ID number	EMPLOY+2

	Marital status	EMPLOY+3
2nd employee	Years service	EMPLOY+4
	ID number	EMPLOY+5

Write a program computing the total years with the company served by the first three employees, storing the answer in LOYAL.

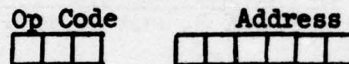
Problem 4.6. The first employee in the previous problem has retired. The employee to replace him has the same marital status, and will be given the same ID number; but he begins with zero years of service. Make the number in EMPLOY+1 a zero.

Only a few pieces of new information remain to be covered. They are all vocabulary words, specifically, the notion of different "fields," and the idea of computer "words" and "records." None of this vocabulary is essential to writing programs, but it helps in talking with other programmers.

So far, all instructions (except HLT) have had two parts—a command and an address. The various parts to an instruction are sometimes called "fields." Commands are written in the "operation code field" (abbreviated Op Code), and addresses are written in the "address field."

<u>Op Code Field</u>	<u>Address Field</u>
CLA	CATS
STR	DOGS

You can think of a field as a set of spaces in an instruction.



The Op Code field has three spaces reserved for the three letters of a command, while the address field has up to six spaces set aside for the letters and numbers of an address.

A "word" is a space set aside in memory for a specific piece of information, such as a man's age, a company's income, or whatever.

A "two-word record" means consecutive pairs of locations in the same address arithmetic series. A "three-word record" was illustrated in Problem 4.5.

Review: Address arithmetic. If a word of memory is addressed by a number, then a series of consecutive addresses can be found by adding the first address to the number, the second to the first, the third to the second, and so on.

PART FIVE: REVIEW AND PRACTICE

This review-and-practice section consists of some previous review boxes, along with a few additional problems.

Names and Places

The computer has two types of locations, the storage (or memory) locations, where numbers are stored; and a work area, where they are added and subtracted. Storage locations can be given names, like COST, NEWARK, and TYPE45, which are called addresses. The central work area is called the accumulator.

Moving Numbers

The CLA command clears out the accumulator and copies in the number from the location addressed. For example, CLA COST clears the accumulator and copies in the number from COST.

The STR command works in reverse, clearing out the location addressed and copying in the number from the accumulator. For example, STR VALUE copies the number from the accumulator into location VALUE, after erasing its previous contents.

Problems. Write them out. Answers are in the separate booklet.

- 5.1. Get pay rate per hour (PAYRT) for computation.
- 5.2. Gross pay has been computed and left in the accumulator. Move it to the gross pay word location (GROPAY).
- 5.3. Relocate an employee's badge number (BADGNO) to location OUTPUT.
- 5.4. Relocate the retirement benefits (RETIRE) to OVTRET.
- 5.5. Relocate SPACE into LOC1, LOC2, and LOC3.

Addition and Subtraction

Numbers can be added or subtracted in the accumulator by the ADD and SUB commands. The answers appear in the accumulator. Numbers in the storage locations are not changed by these commands.

CLA, ADD, and SUB change the accumulator. Only STR can change a storage location. One command changes one number in one place, either the accumulator or a memory location, but not both.

HLT stops the computer, and goes at the end of every program.

Problems. Write them out on a separate sheet.

5.6. A personnel officer determines primary MOS's from each recruit's total score on an Army Classification Battery. There are scores on three tests to be combined for this purpose: spatial perception, verbal, and quantitative. These scores are in SPACE, VERB, and QUANT. They are weighted when combined to give the total score: spatial perception is counted once, verbal twice, and quantitative three times. Prepare a program to obtain a recruit's total score and place the result in memory location MOS1. Don't forget to stop the computer.

5.7. Using the following payroll data, find TOTPAY and NETPAY, but store NETPAY only:

Regular pay (REGPAY) + overtime pay (OVTPAY)
= total pay (TOTPAY).

Total pay (TOTPAY) - deductions (DEDUCT) =
net pay (NETPAY).

5.8. An employee's present salary (PAY) has been doubled. Put the new salary into location PAY and also into location RECORD+18.

Address Arithmetic

Problems. Information for Problems 9-11 follows:

Words of Employee's Record

Location Reserved

Badge number	EMPLOY
Withholding tax	EMPLOY+1
Bond allotment	EMPLOY+2
Hospitalization insurance	EMPLOY+3
Social security	EMPLOY+4
Gross pay	EMPLOY+5
Net pay	EMPLOY+6

5.9. Relocate badge number, gross pay, and net pay to locations relative to OUTPUT. (Badge number goes into OUTPUT, gross pay into OUTPUT+1, and so on.)

5.10. Determine total deductions and put the result into location DEDUCT.

5.11. Determine net pay (gross pay minus deductions) and store in the location reserved for it.

Information for Problems 12 and 13 follows:

EMPLOY	employee badge number
REGTIM	regular time
OVRTIM	overtime
BONUS	bonus
EMPLOY+1	withholding tax
EMPLOY+2	bond allotment
EMPLOY+3	hospitalization
EMPLOY+4	social security
EMPLOY+5	net pay

5.12. In locations REGTIM, OVRTIM, and BONUS are the three factors that make up an employee's gross pay. Compute net pay.

5.13. Write a program to relocate:

badge number to ENNUM.
each gross pay items relative to TOTPAY.
each deduction relative to DEDUCT.
net pay to NETPAY.

5.14. A computer for an artillery battalion maintains a record of the amount of ammunition on hand. Four types of shells are used: high explosive, armor piercing, chemical, and anti-personnel. We know beforehand the quantity of the ammunition which is issued. After a mission the batteries report to Battalion Headquarters the number of each kind of shell fired. Assume the battalion has just finished a firing mission and that the various amounts of each type of shell used have been reported.

<u>Type of Shell</u>	<u>Quantity Issued</u>	<u>Quantity Used</u>
High explosive	AMMO	HEUSED
Armor piercing	AMMO+1	APUSED
Chemical	AMMO+2	CMUSED
Anti-personnel	AMMO+3	PRUSED

Prepare a program that will update the amount of each type of ammunition on hand.

PHASE II

BASIC LOOPING

Everything in this section is aimed at showing how instructions can be repeated automatically.

Often there will be a set of operations you want the computer to perform not just once but many times. For example, you may want to add the number from COST four times. This could be done by writing:

```
CLA COST
ADD COST
ADD COST
ADD COST
```

But in this section you will learn special instructions that can tell the computer, "Add COST four times." This is especially helpful when you want to repeat the same instructions a hundred times, say. Instead of writing the instructions themselves over and over, you need write them only once, along with some additional instructions that say, "Repeat these a hundred times."

Each repetition is called a "loop," which is why this section is entitled "Basic Looping." It is divided into six parts.

Part One: Completing the Loop. This will explain how the computer goes back to begin a set of instructions again.

Part Two: Counting the Loops. Here you will learn how the computer keeps track of the loops.

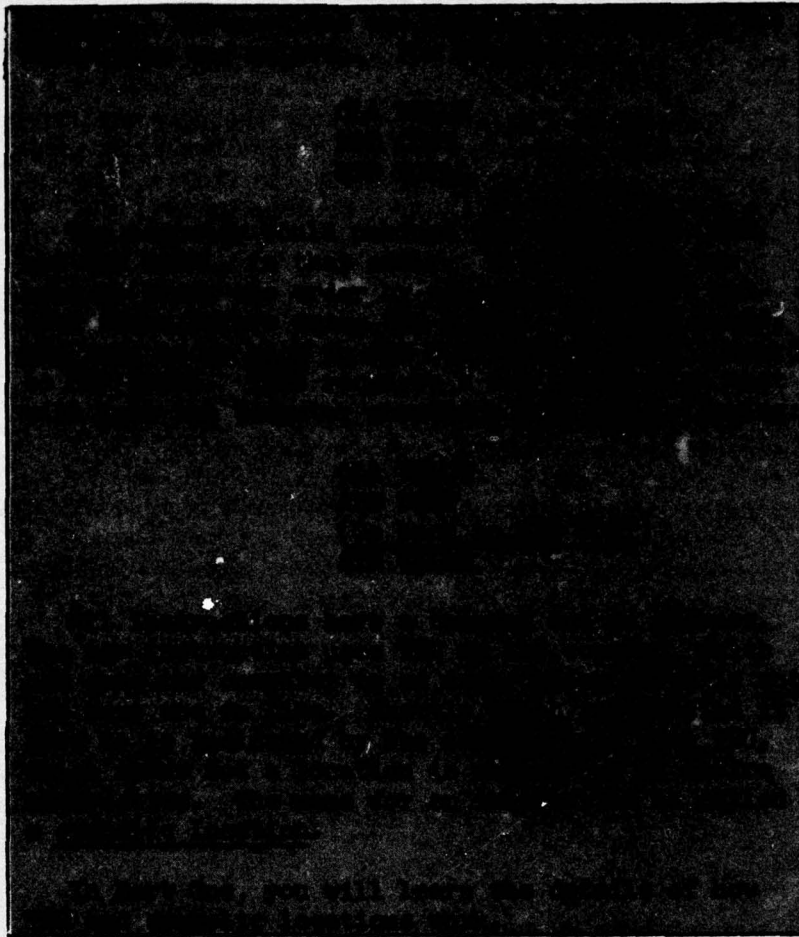
Part Three: Getting Out of the Loop. This part tells how looping is stopped.

Part Four: Looping for Simple Addition. At this point you will start working complete looping problems, beginning with simple addition.

Part Five: Program Preparation. Part Five talks about instructions which get things ready for looping.

Part Six: Review and Practice.

PART ONE: COMPLETING THE LOOP

The TRU Command

address

The command that is used with the new instruction is TRU, which stands for "transfer unconditionally." We will discuss the TRU command below.

TRU is a command, like CLA, ADD, _____, and _____.

Commands tell the computer to do something. CLA says to "clear and add." SUB says to "subtract." TRU says to "go to," "skip to," or "transfer." The TR in TRU, therefore, probably stands for _____.

REPEAT

You have just written ADD COST and now want to transfer (TRU) back to perform the instruction again. That part of the program should look somewhat as follows:

ADD COST
TRU to the instruction "ADD COST"

The TRU command can transfer the program back to an earlier instruction, rather than the next one. In the example above, it could transfer the computer back to do ADD COST again.

TRU has nothing to do with memory locations, like COST or VALUE. It simply tells the computer, "Go to such-and-such a place for your next instruction." TRU says "go to." The name of the place where the next instruction can be found is indicated by the symbolic location used with TRU, which we will take up next.

The new looping instruction uses TRU (transfer unconditionally) as its command. The TRU command can transfer the program out of its normal sequence, sending the computer back to an earlier instruction rather than the one next in line.

Symbolic Locations

TRU says to transfer, but we have not yet discussed how the computer is told where it will find its next instruction. You will now learn that instructions can be given names, or symbolic locations. TRU with a symbolic location tells the computer to transfer to the instruction named by that symbolic location.

Memory locations can be referred to by their names, as you know; you learned long ago that the name of a memory location is called its ____.

Instructions can also be given names. For example, the instruction ADD COST can be named. We'll call it HUBERT. The name of an instruction is written in front: HUBERT ADD COST.

SUB, STR

transfer

REPEAT CLA VALUE. What is the name for the instruction CLA VALUE?

The name of an instruction is called a symbolic location. To review the three kinds of names, in REPEAT CLA VALUE, VALUE is the address, CLA is the _____, and REPEAT is the name for CLA VALUE. It is called a _____.

CLA COST
STR VALUE

A symbolic location can be made up of any combination of letters and numbers, provided the first is a letter, there are no more than six letters and numbers altogether, with no spaces, and all letters are capitalized. Which of the following are not permitted as symbolic locations: HOUSE, A4567, 3MICE, P, SYMBOL, LOOPING?

TRU REPEAT

LEFT

Symbolic locations are used with TRU to indicate where the computer should go for its next instruction.

If ADD COST has been named by writing HUBERT in front of it—HUBERT ADD COST—what instruction is indicated by TRU HUBERT?

Look at the following:

OVER CLA MICE
STR MEN
TRU OVER

What instruction will be performed after TRU OVER?

The name of an instruction is called a _____.

TRU stands for "_____ unconditionally."

TRU HUBERT means "transfer unconditionally to the symbolic location named _____."

What instruction will be performed after TRU REPEAT below?

REPEAT CLA VALUE
TRU REPEAT

ADD COST
STR VALUE

What will transfer the program back to ADD COST below?

ADD COST

AGAIN ADD COST

Symbolic locations have no effect on the instructions they accompany; they act only as names. A computer would read REPEAT CLA VALUE as if REPEAT were not there; only CLA VALUE would actually be performed.

STR VALUE

What are the first two instructions actually performed in the program below?

CLA COST
REPEAT STR VALUE

command,
symbolic location

What instruction would send the computer back to do STR VALUE again in the previous example?

In writing down instructions for your own programs, the commands (CLA, TRU, etc.) are lined up directly under one another, but the symbolic locations are stuck out on the left, as we have been doing. Which of the following pairs of instructions are lined up correctly, the pair on the left or the pair on the right?

3MICE,
LOOPING

REPEAT CLA VALUE
TRU REPEAT

REPEAT CLA VALUE
TRU REPEAT

ADD COST

Instructions can be named by writing a symbolic location in front, such as REPEAT in REPEAT ADD COST. The symbolic location can be used later with the TRU command to name that instruction as the next one to perform. TRU REPEAT tells the computer to go back and do ADD COST.

CLA MICE

symbolic location

transfer

You now have all the basic information on "Completing the Loop." All that remains is to practice, review, and try some problems on your own.

Questions below are based on the following:

HUBERT

REPEAT ADD COST
STR VALUE
TRU REPEAT

CLA VALUE

The first two instructions performed by the computer are _____ and _____.

The instruction performed immediately after TRU REPEAT is _____.

TRU AGAIN

Since the computer normally performs instructions in order, the instruction after the second ADD COST would be _____.

TRU REPEAT is next in order after STR VALUE, so the computer would go back and perform _____ a third time.

And that's looping. Don't worry about how to stop looping for now. We will discuss that later on.

Review. To repeat a set of instructions requires two things: (1) a command that says "Repeat" or "Transfer" to an earlier instruction and (2) a way of identifying that instruction. The command that we used to break the normal sequence of instructions is TRU, which stands for "Transfer Unconditionally." The instruction to which we refer is transferred to now by putting a symbol location in front of that instruction, and later using the symbolic location with the TRU command. TRU REPEAT tells the computer to transfer to the instruction noted above. The two features taken together, TRU and a symbolic location, allow the computer to complete the loop.

accumulator,
COST

Write out the following problems on a separate piece of paper. Remember, answers are in the separate booklet. Use them only if you have to.

Problem 1.1. Write a program adding the cost of one vacuum tube to the cost of all transistors. The cost of a vacuum tube is in COST. The cost of a transistor is in TRANS. Use REPEAT as the symbolic location for looping. The program should give the sum of COST+TRANS+TRANS+TRANS+TRANS+etc.

Problem 1.2. Write a program adding the cost of one pair of pants to the cost of all suits on hand. The cost of one pair of pants is in PANTS. The cost of a suit is in SUITS. Use REPEAT as the symbolic location for looping.

Problem 1.3. Write a program that will subtract the number 1 from the number in COUNT, storing the answer back in COUNT. Then have the program loop back to repeat the process. The number 1 is in location ONE. Use AGAIN as the symbolic location for looping. The number in COUNT should be reduced by 1 on each loop.

ADD COST

Problem 1.4. Write a program that will reduce the number in LOOPS by 1, putting the new number back into LOOPS. Have the program loop back to repeat the process over and over, using AGAIN as the symbolic location for looping. The number in LOOPS should be smaller by 1 on each loop. The number 1 is in location ONE.

The CLA ZRO Instruction

You have been taught to clear the accumulator on the first instruction of any program by writing CLA as the first command.

To review, CLA COST erases the number in the _____ and replaces it with the number from _____.

If there is a 6 in the accumulator and a 4 in COST, the number in the accumulator after CLA COST will be _____.

Now let's say you want to add the number from COST many times using a looping program. You don't want to repeat CLA COST over and over, since each CLA erases all previous additions, leaving only one COST no matter how many times the instruction is repeated.

The instruction you want to repeat is ADD COST, but you want to be sure the accumulator is cleared, or zeroed, before you start.

The solution is to clear the accumulator by writing CLA ZRO as the first instruction.

CLA ZRO says, "Clear the accumulator and copy in a zero."

Notice how "zero" is spelled in the instruction. It is CLA _ _ _.

The looping program to add the number from COST many times would be:

```

          CLA
REPEAT   ADD COST
          TRU REPEAT

```

COUNT

The accumulator can be zeroed in preparation for a looping program by writing CLA ZRO.

subtracting

CLA COUNT
SUB ONE

The following problems test your understanding of Part One: Completing the Loop. Write them out.

Problem 1.5. Write a program to compute the cost of all vacuum tubes in stock. The cost of one tube is in COST. Use REPEAT as the symbolic location for looping.

No

STR COUNT

Problem 1.6. Write a program to compute the cost of all hats in stock. The cost of one hat is in HAT. Use AGAIN as the symbolic location for looping.

Problem 1.7. Write a program to compute the cost of all sets of hats, coats, and gloves in stock. The cost of one of these items is in HAT, COAT, and GLOVE, respectively. Use LOOPER for looping.

PART TWO: COUNTING THE LOOPS

COUNT

two

Rockets are counted down to zero: 3-2-1-0. We will do the same in counting loops; we will count them down to zero.

ZRO

Say the location containing the number of loops desired is in COUNT. You can count the loops as they are performed by "counting down" the number in ____.

To count down means (adding or subtracting?) ____ a 1 from COUNT each time a loop is completed.

Storage location ONE contains the number 1. What two instructions will subtract a 1 from COUNT?

Recall that addition and subtraction take place in the accumulator, not in the memory locations. Is the number in COUNT changed by the instructions CLA COUNT and SUB ONE?

A third instruction is needed to get the changed number back into COUNT. What is it?

Here's a desk check of the three counting instructions. Assume there is a 5 in COUNT.

CLA COUNT The accumulator is cleared and the 5 from COUNT is copied in. It is copied, not changed.

SUB ONE The number in the accumulator is reduced by 1, making it a 4.

STR COUNT The 5 already in COUNT is erased and a 4 is copied in. COUNT has now been changed from 5 to 4.

The location containing the number of loops required is called the loop counter. In previous examples, the loop counter was location ____.

If the three counting instructions are made part of the loop, they will be performed automatically, and the loops will count themselves, as in the following example:

```
REPEAT  CLA COUNT
        SUB ONE
        STR COUNT
        TRU REPEAT
```

If COUNT contains a 2, it will be "counted down" to zero after how many loops?

If you only wanted to perform two loops, you would want the computer to stop when the number in COUNT was reduced to ____.

To review, the memory location containing the number of loops you want completed is called the ____.

In previous examples, the memory location named ____ was the loop counter, since that location contained the number of loops desired.

In practice, however, the loop counter will be different for each problem. For example, say you want to subtract THESE from THEM as many times as THOSE. Which is the loop counter?

The following example shows how the counting instructions work in a looping program. PASSES (the loop counter) starts with a 2.

	<u>First Loop</u>	<u>Second Loop</u>
REPEAT CLA PASSES	Bring up the 2.	Bring up the 1 now in PASSES.
SUB ONE	Reduce it to 1.	Reduce it to 0.
STR PASSES	<u>Erase the 2 already in PASSES</u> and copy in the 1 from the accumulator.	Erase the 1 in PASSES and substitute the 0 from the accumulator.
TRU REPEAT	Go back to CLA PASSES.	

The loop counter is the name of the location containing the number of loops needed, not the number itself.



Write out the following problems on a separate piece of paper. Answers are in the separate booklet.

zero

loop counter

Problem 2.1. You are working on a long looping program where the loop counter is COUNT. Write out the three counting instructions that you would insert into this program.

COUNT

THOSE

Problem 2.2. You are working on a long program in which you want to complete as many loops as the number in LOOPER. Write out the loop counting instructions you would insert into the complete program.

Problem 2.3. Write a simple looping program that will repeat the three counting instructions, using CASE as the loop counter and LOWER as the symbolic location for looping.

Problem 2.4. Write a program that will repeatedly add a 1 to CARD and subtract a 1 from FILE, using AGAIN as the symbolic location.

Problem 2.5. Write a program that will subtract a 2 from COUNT on each loop. There is a 1 in ONE. Use REPEAT as the symbolic location for looping.

PART THREE: GETTING OUT OF THE LOOP

The TRZ Command

You will want to stop looping when the number in the loop counter (the number of loops required) reaches zero. Thus, you need an instruction that transfers the program out of the loop when the number in the accumulator is zero (for example, when COUNT minus ONE equals zero). The TRZ command is used with that instruction. You will now learn how it works.

TRZ is another command, like CLA, ADD, SUB, STR, HLT, and _____.

STR UNIT,
CLA ONE

TRZ asks whether the number in the accumulator is a zero. You can guess, therefore, that the TR stands for _____ and the Z stands for _____.

Notice that TRZ asks its question of the accumulator and not of a memory location.

If the number in the accumulator is a zero, the computer will transfer to the symbolic location indicated. Here's a trivial example: What will be performed after TRZ REPEAT below?

HLT

```
REPEAT  CLA ZRO
          TRZ REPEAT
```

If the accumulator does not contain a zero, the computer will not transfer, but will perform the next instruction in line instead.

- 37 -

COUNT contains a 1. What instructions will be performed after TRZ REPEAT below?

```
REPEAT  CLA COUNT
        SUB ONE
        TRZ REPEAT
```

Here's a desk check of that problem; COUNT = 1.

```
REPEAT  CLA COUNT  Copy a 1 into the accumulator.
        SUB ONE    1 - 1 = 0. There is a zero in
                   the accumulator.
        TRZ REPEAT The TRZ command asks if there is
                   a zero in the accumulator. Yes,
                   there is. So transfer to symbolic
                   location REPEAT.
```

With the TRU command, the program always transfers. But with the TRZ command, the program transfers only if the accumulator has a zero. That's what TRZ stands for—transfer if zero.

Look at the difference between the two examples below: one uses TRZ, the other uses TRU.

```
REPEAT  CLA ONE
        TRZ REPEAT
        STR UNIT
```

```
REPEAT  CLA ONE
        TRU REPEAT
        STR UNIT
```

What instruction would come after TRZ REPEAT in the program on the left? After TRU REPEAT on the right?

TRU

transfer,
zero

If the accumulator does not contain a zero, as in the example above, TRZ tells the computer to pass on to the next instruction, just as if TRZ weren't even there.

If location COUNT contains the number 1, in the example below, what instruction will be performed after TRZ STOP-- CLA COUNT or HLT?

```
MULT  CLA COUNT
      SUB ONE
      STR COUNT
      TRZ STOP
      TRU MULT
      STOP  HLT
```

CLA ZRO

Transfer commands—TRU and TRZ—do not change numbers. Whatever was in the accumulator just before a transfer command will still be there for the next instruction.

For example: CLA COUNT
 SUB ONE
 TRZ STOP
 STR COUNT

If COUNT = 2, the instructions will be carried out just as if TRZ STOP were not there. COUNT minus ONE will be stored back into COUNT.

The TRU command works the same way. What gets stored in TOTAL below-- COST, VALUE, or TOTAL?

SUB ONE

 CLA COST
 TRU FINAL
 CLA VALUE
FINAL STR TOTAL

once

COST is copied into the accumulator and stays there during the transfer to symbolic location FINAL. What-
ever is in the accumulator stays there while the program transfers.

zero

The TRZ command transfers the program to the symbolic location indicated only if the accumulator contains a zero. Otherwise, the next instruction is performed. TRZ FINAL says to transfer to FINAL only if the accumulator is zeroed.

Write out the following problems on a separate piece of paper. Answers are in the separate booklet.

Problem 3.1. You are working on a long looping program where the loop counter is COUNT. Write the three counting instructions you would use, plus a fourth instruction to transfer the program to the symbolic location named STOP when the loop counter reaches zero.

Problem 3.2. You are working on a long looping program where the loop counter is LOOPS. Write the three counting instructions you would use, plus a fourth instruction to transfer the program to STOP when the loop counter reaches zero.

Symbolic Location STOP

The symbolic location STOP is used only to stop the computer. Thus, when the program transfers to STOP, it must find a HLT command. Usually, a transfer instruction—TRZ STOP or TRU STOP—will be written inside the program, with the symbolic location STOP at the very end naming the instruction HLT.

Questions below are based on the following instructions, where location ONE contains a 1:

```

                CLA ONE
LOOP   TRZ STOP
                SUB ONE
                TRU LOOP
STOP   HLT

```

What will be performed after TRZ STOP on the first loop--SUB ONE or HLT?

COST

What number is in the accumulator immediately after TRU LOOP, before the TRZ STOP instruction is performed a second time?

How many times will TRU LOOP be performed?

Here's a desk check of that problem:

CLA ONE	Copy a 1 into the accumulator.
LOOP TRZ STOP	Does the accumulator have a zero? No, it's a 1. So don't transfer, but move on. <u>The 1 stays in the accumulator.</u>
SUB ONE	$1 - 1 = 0$. The accumulator has a zero after this instruction.
TRU LOOP	Transfer to LOOP. <u>The zero stays in the accumulator.</u>
---	---
LOOP TRZ STOP	Starting the loop again. Does the accumulator have a zero? Yes, then transfer to STOP; TRZ STOP says, "Transfer if zero to STOP."
---	---
STOP HLT	The command at STOP is HLT, so the computer halts after one TRU LOOP.

The TRZ STOP instruction will halt the computer by transferring to STOP HLT if the accumulator has a zero; otherwise, the program will not transfer, but will go on to the next instruction. Neither of the transfer commands—TRZ and TRU—change numbers in the accumulator.

Write out the following problems on a separate piece of paper. Again, the answers are in the Answer Booklet.

Problem 3.3. A supply sergeant is making his annual inventory. The number of hats he has left from last year is in HATS1. Copy the number into the location for this year, HATS2, if there are any left. Otherwise, stop the computer.

Problem 3.4. Write a program to add a 1 to FILE and subtract a 1 from CARD, stopping the computer when CARD is down to zero. Use REPEAT for looping.

The Test for Completion

The three instructions for counting the loops can be put together with the TRZ STOP instruction to stop looping when the loop counter has been reduced to zero. The complete block of four instructions is called the test for completion, since it checks whether all the required loops have been made. We will show how it works below.

Assume COUNT holds the number of loops required; COUNT, therefore, is the loop counter. When it has been reduced to zero we will want to stop looping. This can be guaranteed by adding a TRZ STOP instruction.

The complete test for completion would appear as follows:

```
CLA COUNT
SUB ONE
STR COUNT
TRZ STOP
```

We will now make them part of a complete looping program and review how they work.

Assume COUNT contains a 2. That is, we want to stop looping after two repetitions.

```
LOOP  CLA COUNT  Copy the 2 into the accumulator.
      SUB ONE    Subtract 1, making it 1.
      STR COUNT  Copy the 1 into COUNT. A 1 remains
                        in the accumulator.
      TRZ STOP   Transfer to STOP if the accumulator
                        has a zero. It does not, so move on.
                        (continued on next page)
```

TRU LOOP	Transfer to LOOP.
STOP HLT	The computer has transferred back up, so it will not reach this instruction after the first loop.

LOOP CLA COUNT	The 1 from COUNT is copied in.
SUB ONE	1 - 1 = 0.
STR COUNT	Copy the zero into COUNT. <u>A zero remains in the accumulator.</u>
TRZ STOP	Does the accumulator have a zero? <u>Yes</u> , so transfer to STOP after two loops.

STOP HLT	

If the loop counter contains the number of required loops, the test for completion will insure that looping stops after all loops have been made.

The test for completion is a group of four instructions that counts the loops and transfers the program out of the loop when all loops have been completed. Its basic form is:

1. CLA the loop counter
2. SUB the number 1
3. STR back in the loop counter
4. TRZ someplace outside the loop

Write out the following problems on a separate piece of paper. Answers are in the Answer Booklet.

Problem 3.5. Write a looping program to repeat the test for completion, using MANY as the loop counter and DOWN as the symbolic location for looping. Location ONE contains the number 1.

Problem 3.6. Write a looping program to repeat the test for completion, using DECK as the loop counter and LOWER for looping. Location UNIT contains the number 1.

PART FOUR: LOOPING FOR SIMPLE ADDITION

COST



The solution lies in adding the number to the same location on each loop. The number itself doesn't change, but the location for the answer keeps increasing.

Incidentally, one completion of a set of looping instructions is called a "pass through the loop." The second time through would be called the second pass.

two

Say you want to add the number from COST many times, storing each increase in TEMP. At the end of looping, TEMP should equal COST+COST+COST+COST, and so on, for as many COSTs as are desired.

The heart of the program is as follows:

```
REPEAT  CLA COST
          ADD TEMP
          STR TEMP
          TRU REPEAT
```

Before we begin the explanation, it's important to make clear which location contains the number to be added and which one is used for the answer. COST is to be added, and TEMP is the answer location.

Look back at the basic program. Notice that the answer location TEMP is used twice:

TEMP

```
CLA COST
  TEMP
  TEMP
```

First, assume TEMP starts with a zero. If so, what is in TEMP after the three instructions below have been completed--zero or COST?

```
CLA COST
ADD TEMP (It contains a zero.)
STR TEMP
```

TEMP would contain COST. The first two instructions add COST to zero, giving simply COST. The third instruction, STR TEMP, puts COST into TEMP.

So, after the three instructions have been performed once, one COST is in TEMP.

If the instructions were executed a second time, TEMP would start with one COST already in it. What would then be in TEMP at the end of the three instructions (shown again below)--one COST or two?

```
CLA COST
ADD TEMP (It contains one COST.)
STR TEMP
```

TEMP would contain two COSTs. CLA COST puts one COST in the accumulator. ADD TEMP adds a second COST, since TEMP already contains one COST. STR TEMP then puts both COSTs into TEMP.

After the instructions are completed a second time, TEMP contains two COSTs.

In short, each time the instructions are performed, another COST goes into TEMP.

This means the contents of TEMP change each time, increasing by the amount in COST with each pass.

If you want to get the sum of COST+COST+COST, etc., into TEMP, this is the way to do it:

```
REPEAT CLA COST
      ADD
      STR TEMP
      TRU REPEAT
```

ADD
STR

The basic format for the adding instructions:

1. clear-and-add the number to be added;
2. add the contents of the answer location;
3. erase the answer location and put in the new sum.

You are probably having a little trouble understanding how the three adding instructions work. We will try to explain them differently.

SHOE

If PRICE contains the number to be added repeatedly, and you have chosen TOTAL as your answer location, the diagrams below trace out what happens on the first two passes. They assume that TOTAL starts with a zero.

<u>Instruction</u>	<u>PRICE</u>	<u>Accumulator</u>	<u>TOTAL</u>
CLA PRICE	PRICE	PRICE	
ADD TOTAL		+ 0	zero
STR TOTAL		PRICE	PRICE
CLA PRICE	PRICE	PRICE	
ADD TOTAL		+PRICE	PRICE
STR TOTAL		2 PRICES	2 PRICES

LACE + 4

Notice the second pass, particularly. If there is one PRICE in TOTAL and two PRICES in the accumulator, how many PRICES will be in TOTAL after the instruction STR TOTAL--one or two?

There will be two, because the STR command erases the location addressed, erasing the one PRICE before copying in the contents of the accumulator, which is two PRICES.

ZRO

The general rule for repeated addition is: Add the number to the answer location, putting the sum back into the answer location.

Now try a few problems. They are designed to give you experience in distinguishing the number to be added from the answer location, and writing the three adding instructions.

Problem 4.1. You want to increase MANY by an amount equal to ONCE on each loop. Write the three adding instructions.

Problem 4.2. You want to make location SHOE equal the sum of LACE+LACE+LACE+LACE, etc. Write the three adding instructions.

VALUE

The adding instructions can be made into a looping program very easily. For example, Problem 4.2 could be made to loop as follows:

```
AGAIN CLA LACE
      ADD SHOE
      STR
      TRU AGAIN
```

The program has a big defect, however. The first pass should put one LACE into SHOE. That is, SHOE should equal LACE after the first pass: $SHOE = LACE$.

However, SHOE will equal LACE only if SHOE contains a zero at the very beginning. What will SHOE contain after the first pass if it starts with a 4; will $SHOE = LACE$ or will $SHOE = LACE + 4$?

If the answer location does not start with a zero, the program will not give the correct answer. It will give the sum of all additions plus what was in the answer location before looping began.

Therefore, the answer location must be zeroed at the start of the program.

two

The business of zeroing a location is called "cleaning out garbage," since it cleans out any previous material, changing the contents to zero.

If VALUE is chosen as the answer location, the way to clean out garbage from that location is:

```
CLA
STR VALUE
```

You should start to think of programs in terms of their component parts rather than in terms of individual instructions.

The problems to follow can be broken down into three parts: zeroing the answer location (2 instructions), adding (3 instructions), and a final instruction for looping, for a total of 6 instructions.

Here's an example. A hardware company wants to compute the total value of all hammers in stock. The price of one hammer is in HAMMER. The total value is to go into VALUE. LOOPER can be used for looping.

What is the answer location?

What instructions will clean out the garbage from that location? These instructions make up the first part of the program. Think of them as a single block.

The company wants the sum of HAMMER+HAMMER+HAMMER+HAMMER, and so on, with each increase going into VALUE. What are the three adding instructions? They make up the second block of instructions; think of them as a unit.

Finally, you will need an instruction to go back and do another addition. What is that instruction? Call it the looping instruction and think of it as the final part of the program.

The complete program will have six instructions but only three parts:

	<div>CLA ZRO STR VALUE</div>	Part 1—cleaning out garbage
LOOPER	<div>CLA HAMMER ADD VALUE STR VALUE</div>	Part 2—adding
	<div>TRU LOOPER</div>	Part 3—looping

Problem 4.3. A university needs a program to compute the money spent on athletic scholarships last year. Each athlete made the same money, the figure in PAYOFF. Put the answer in TOTAL. Use REPEAT for looping.

Problem 4.4. Write a program to compute the value of all suits in stock. The price of one suit is in SUIT. Put the total in VALUE. Use REPEAT for looping.

Now, to make a complete program you need only insert a test for completion between the adding and looping instructions (including a STOP HLT at the very end).

CLA ZRO
STR VALUE

symbolic location

(2 instructions)

(3 instructions)

(4 instructions)

TRU LOOPER

(1 instruction)

STOP HLT

Answer the questions on the following example.
You need not write the program itself.

First, let's get straight on what is required.

We will assume that each car is valued at \$4,000. The heart of the program should do the following, in the order given:

1. add in \$4,000 (the cost of one car),
2. check off one addition,
3. go back and add again.

Now we will set up the blocks of instructions that accomplish these functions.

First, the location used for the sum of all additions must be zeroed out. Otherwise, the answer will be too big. What is that location?

The first block of instructions should clean out the garbage from that location. What are those instructions?

Next, we need the instructions that will add in the cost of one car to the location for the total of all additions. Where is the cost of one car?

What are the three basic adding instructions?

Next, we need some instructions that will check off the fact that an addition has been performed. How many instructions are required? What location contains the number to be used for this check?

That location is the loop counter, used in the test for completion. The test for completion checks off each addition, just after it has been performed.

What are the four instructions which make up the test for completion in this example?

Don't forget that the last instruction in the test for completion uses TRZ, not TRU. TRZ says, "Transfer if zero." So the computer will transfer to STOP when the loop counter is down to zero, indicating no more additions need be done.

Finally, you need an instruction to tell the computer it's time to repeat the loop. What is that instruction?

To review, programs for simple addition require only a few basic blocks of instructions:

1. clean out garbage from the answer location
2. the basic adding instructions
3. the test for completion to count each addition as it is performed
4. a looping instruction to add again
5. STOP HLT

For the sample program you just worked on, the breakdown by blocks would look as follows:

	CLA ZRO STR TOTAL	Cleaning out garbage	TOTAL
REPEAT	CLA AUTO ADD TOTAL STR TOTAL	Addition	
	CLA STOCK SUB ONE STR STOCK TRZ STOP	Test for completion	AUTO
	TRU REPEAT	Looping	
	STOP HLT		3, STOCK

CLA ZRO
STR TOTAL

CLA AUTO
ADD TOTAL
STR TOTAL

CLA STOCK
SUB ONE
STR STOCK
TRZ STOP

Problem 4.5. A payroll officer needs a program to compute the total amount paid last month to privates. A private's salary is the number in SALARY. The number of privates paid is in MEN. Store the answer in TOTAL. Use REPEAT for looping.

Problem 4.6. A publishing company wants a program to compute the sales of a certain book. The price of the book is in PRICE. The number of books sold is in SALES. Store the answer in GROSS. Use AGAIN for looping.

It is easy to confuse the loop counter, used in the test for completion, with the location containing the number to be used for repeated addition.

Just ask yourself, "What number do I want to add?" Use that location for the adding instructions.

Then ask yourself, "How many times do I want to add this number?" Use that location as your loop counter in the test for completion.

TRU REPEAT

Review. The looping procedure for simple circuits and the basic procedure for general circuits are not to know a procedure for each circuit. The main idea in the looping theory is to use a loop, then solve by an appropriate method the problem. The looping method is a special case of the general method, which is a special case of the general method. The answer is not in the looping theory, but in the

none

Problem 4.7. Write a program to compute the total value of all pencils in a warehouse and store the answer in VALUE. The number of pencils is in PENCIL. All pencils cost the same, the number in PRICE. Location K1 contains a 1. Use REPEAT as the symbolic location for looping. Ask yourself, "What number do I want to add?" Use that location in the adding instruction. Then ask, "How many times?" Use that location in the test for completion.

zero

Problem 4.8. A personnel officer wants to figure the total leave time all men in the battalion will have next month. Each man is allotted three days; the number 3 is in THREE. The number of men in the battalion is in MEN. Put the total leave in location ALL. Use MULT for looping. Location UNIT contains a 1.

COUNT
TRZ STOP

PART FIVE: PROGRAM PREPARATION

Preview. Only a few points remain to be covered. All of them deal with instructions that should be performed before looping begins. Hence, they are called program preparation. The instructions

1. check that the loop counter holds a number greater than zero;
2. save that number by storing it in a temporary location; and
3. zero locations to be used in the program, called "cleaning out garbage."

Checking the Loop Counter

Most of the time the programmer does not know the number of loops that should be performed. He knows where the number is located (in the loop counter), but he doesn't know what that number is. To guard against the possibility that there are no loops (zero loops) to be performed, he writes as his first two instructions:

CLA the loop counter
TRZ STOP

A company wants to know how much money is paid annually to its secretaries. All secretaries make the same wage; that figure is in SALARY. The number of secretaries is in TYPIST. The program should add the number in _____ as many times as the number in _____. Which is the loop counter?

Assume all secretaries have been fired. The number in the loop counter would then be _____.

How many additions would you want to perform?

If there is a zero in the loop counter, there are no loops at all to do and we may as well stop right at the very beginning.

If COUNT is the loop counter, the first two instructions in the program should be:

CLA _____
____ _

If there are no items to count, add, or otherwise process, there will be a zero in the loop counter. Thus, the first step in program preparation is to check the loop counter for zero, transferring the program to STOP if it should have a zero.

Saving Numbers

The test for completion lowers the loop counter by 1 for each completed loop. This means the original number is destroyed. It can be saved during program preparation by copying it into a temporary location and using the temporary location in the test for completion rather than the loop counter itself. This way the original number can be saved and used again:

CLA the loop counter
STR it into a temporary location

If the loop counter is counted down for each completed loop, what number does it contain when the TRZ STOP instruction stops the computer?

The entire program is then useless. If you tried to use it again, the loop counter would start with a zero and the computer would not know how many loops to perform; it would start counting down from zero rather than from the desired number.

This means the loop counter must be saved during program preparation. This is accomplished by copying the number into a temporary location and using that location rather than the loop counter itself in the test for completion.

If the loop counter is COUNT, and you copied its contents into TEMP, which location should be used in the test for completion?

What instructions will copy the loop counter PASSES into temporary location NUMBER?

We can combine the program preparation steps which (1) check the loop counter for zero, and (2) save that number, by recalling that transfer commands do not change numbers in the accumulator.

Look at the following:

CLA COUNT
TRZ STOP
STR TEMP

If COUNT contains a 5, what is in the accumulator after TRZ STOP? What is stored in TEMP?

If COUNT is the loop counter, those three instructions will check it for zero and save it by copying it into TEMP.

In general, the program preparation instructions that (1) check the loop counter for zero, and (2) copy it into a location to be used in the test for completion will be like the following:

1. clear-and-add the loop counter
2. TRZ STOP (to check for zero)
3. store it in a temporary location

In the following problems, write out only the three program preparation instructions we have just learned plus the test for completion. The problems are designed to give you practice in identifying the loop counter, checking it for zero, storing it, and using the temporary location, rather than the original number, in the test for completion.

zero

TEMP

Problem 5.1. A furniture company had poor luck selling a certain type of sofa last year. They carry three types, whose selling prices are in locations SOFA1, SOFA2, and SOFA3. The low sales were from the second type. Write the program preparation instructions that will check whether any of that type were sold. The number of sofas sold for each of the three types of sofa is in SOLD1, SOLD2, and SOLD3, respectively. COUNT is available for temporary storage. Write the block of instructions making up the test for completion for a program computing total sales for the second type of sofa.

CLA PASSES
STR NUMBER

5, 5

Problem 5.2. An insurance company sells two kinds of insurance, accident and life insurance. Yearly premiums on each are in ACC and LIFE. Numbers of persons paying premiums for each type last year are in locations SLIP and AGE, respectively. Temporary locations for each of these numbers are FALL and OLD. Write the three program preparation instructions for a program to compute the total amount of money paid last year for life insurance. Then write the test for completion instructions for the same program.

Cleaning Out Garbage

The final point to cover under program preparation is something we have already discussed, the business of cleaning out garbage from answer locations.

If the location where the sum of each addition is stored contains anything but a zero before the first pass, the program will give the wrong answer. It will give the sum of all desired additions plus whatever was in the answer location at the start.

We must guarantee that the location used for the answer has a zero to begin with.

As you know, this is done as follows:

CLA ZRO

STR in the location for the answer

Incidentally, those two instructions also zero the accumulator, since the STR command only copies the number already in the accumulator. Since a zero is in the accumulator from CLA ZRO, it stays there for the first looping instruction.

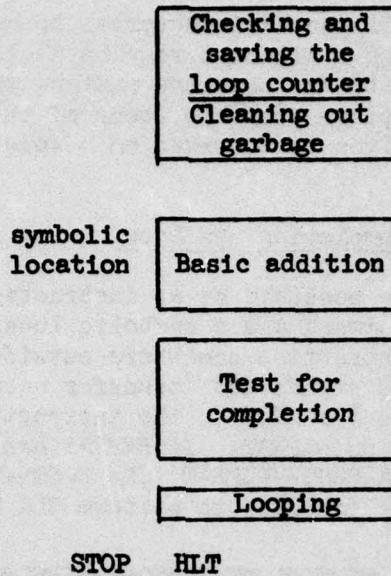
This is necessary in those situations (which we will run into later) where the first instruction in the loop uses an ADD command, rather than CLA.

Review. The program preparation instructions are as follows:

1. CLA ZRO
2. STR in the location for the answer
3. CLA ZRO
4. STR in the location for the loop counter
5. CLA ZRO
6. STR in the location for the loop counter
7. CLA ZRO
8. STR in the location for the loop counter
9. CLA ZRO
10. STR in the location for the loop counter
11. CLA ZRO
12. STR in the location for the loop counter
13. CLA ZRO
14. STR in the location for the loop counter
15. CLA ZRO
16. STR in the location for the loop counter
17. CLA ZRO
18. STR in the location for the loop counter
19. CLA ZRO
20. STR in the location for the loop counter
21. CLA ZRO
22. STR in the location for the loop counter
23. CLA ZRO
24. STR in the location for the loop counter
25. CLA ZRO
26. STR in the location for the loop counter
27. CLA ZRO
28. STR in the location for the loop counter
29. CLA ZRO
30. STR in the location for the loop counter
31. CLA ZRO
32. STR in the location for the loop counter
33. CLA ZRO
34. STR in the location for the loop counter
35. CLA ZRO
36. STR in the location for the loop counter
37. CLA ZRO
38. STR in the location for the loop counter
39. CLA ZRO
40. STR in the location for the loop counter
41. CLA ZRO
42. STR in the location for the loop counter
43. CLA ZRO
44. STR in the location for the loop counter
45. CLA ZRO
46. STR in the location for the loop counter
47. CLA ZRO
48. STR in the location for the loop counter
49. CLA ZRO
50. STR in the location for the loop counter
51. CLA ZRO
52. STR in the location for the loop counter
53. CLA ZRO
54. STR in the location for the loop counter
55. CLA ZRO
56. STR in the location for the loop counter
57. CLA ZRO
58. STR in the location for the loop counter
59. CLA ZRO
60. STR in the location for the loop counter
61. CLA ZRO
62. STR in the location for the loop counter
63. CLA ZRO
64. STR in the location for the loop counter
65. CLA ZRO
66. STR in the location for the loop counter
67. CLA ZRO
68. STR in the location for the loop counter
69. CLA ZRO
70. STR in the location for the loop counter
71. CLA ZRO
72. STR in the location for the loop counter
73. CLA ZRO
74. STR in the location for the loop counter
75. CLA ZRO
76. STR in the location for the loop counter
77. CLA ZRO
78. STR in the location for the loop counter
79. CLA ZRO
80. STR in the location for the loop counter
81. CLA ZRO
82. STR in the location for the loop counter
83. CLA ZRO
84. STR in the location for the loop counter
85. CLA ZRO
86. STR in the location for the loop counter
87. CLA ZRO
88. STR in the location for the loop counter
89. CLA ZRO
90. STR in the location for the loop counter
91. CLA ZRO
92. STR in the location for the loop counter
93. CLA ZRO
94. STR in the location for the loop counter
95. CLA ZRO
96. STR in the location for the loop counter
97. CLA ZRO
98. STR in the location for the loop counter
99. CLA ZRO
100. STR in the location for the loop counter

Write out programs for the following problems, including five program preparation instructions. The only things new are (1) three new instructions to check and save the loop counter, and (2) use of the temporary location for the loop counter in the test for completion.

The general format for the programs is:



Problem 5.3. Write a program to compute the total value of all 6SN7 vacuum tubes in stock, placing the answer in VALUE. COST contains the value of a single tube, ONE contains the number 1, and COUNT contains the number of vacuum tubes on hand. Use TEMP for temporary storage of COUNT, and REPEAT as the symbolic location for looping.

Problem 5.4. A manufacturer makes three kinds of towels—face, hand, and bath towels. Their prices are in FACE, HAND, and BATH, respectively. The number sold of each type is in FACE+1, HAND+1, and BATH+1. Temporary locations for each type are available in FACE+2, HAND+2, and BATH+2. The manufacturer wants to know the gross sales of hand towels. Put the total sales in HAND+3. Use WASH for looping. There is a 1 in location KON1.

PART SIX: REVIEW AND PRACTICE

You now have all the basic material to be covered under Basic Looping. All that remains is to review the important points and practice putting them together into complete programs, some of them a little longer than those you have worked on before.

Completing the Loop

Looping is made possible by an instruction which uses TRU as its command and a symbolic location, which names an instruction somewhere outside the normal order. TRU stands for "transfer unconditionally." TRU REPEAT says to transfer to the instruction named by symbolic location REPEAT. If REPEAT had been placed next to CLA COST—REPEAT CLA COST—TRU REPEAT would instruct the computer to perform CLA COST next.

Problem 6.1. Program A below says exactly the same thing as one of the other programs—B or C. Which one?

<u>Program A</u>	<u>Program B</u>	<u>Program C</u>
TRU FAIL	ADD CON5	SUB CON5
PASS ADD CON5	SUB CON5	ADD CON5
TRU STOP	HLT	HLT
FAIL SUB CON5		
TRU PASS		
STOP HLT		

The Test for Completion

The test for completion is a group of four instructions that counts the loops and transfers the program out of the loop when all loops have been completed. Its basic form is:

1. clear-and-add the number in the loop counter
2. reduce it by one for the loop just completed
3. put the lowered number back in the loop counter
4. transfer out when the number (which is still in the accumulator) is down to zero, using TRZ (transfer if zero)

Problem 6.2. A publisher of encyclopedias sells its books in sets of two, the encyclopedia itself and a separate index. The prices are different since the books are of different sizes. The price of the encyclopedia is in BIG and the price of the index is in LITTLE. The company has had a good year and they would like to know the total value of all sets sold, placing the answer in TOTAL. Write a program to compute the value of sets sold. The number of sets sold is in SOLD. Location TEMP is available for temporary storage. Use SELL for looping. Notice that the problem requires repeated addition, not just of one number, but two.

Program Preparation

Program preparation instructions act to check loop counters for zero and, if they are greater than zero, save them. Other instructions zero out answer locations, if this is called for.

Saving the loop counter is basically a matter of relocating a number (after checking for zero). Sometimes it is necessary to save other numbers as well. This is done in the same way, by copying the number into a temporary location during program preparation and using the temporary location in later instructions.

The problem below requires you to save not only the loop counter but another number as well. Program preparation will be a little longer. Everything else is the same as usual.

Problem 6.3. A retail house for musical instruments updates its records at the end of each month. Last month they added a new item, a supply of post horns, valued at a certain amount, which is contained in location HORNS. The number of post horns sold is in SOLD. The value of one horn is in PRICE. Write a program to compute the value of post horns sold, putting the total in HORNS. Save the original number in HORNS by storing it in BLOW. COUNT is available for temporary storage of the loop counter. Use POST for looping.

Simple Addition and Subtraction

The basic format for the adding instructions is as follows:

1. clear-and-add the number to be added
2. add the contents of the answer location
3. erase the answer location and store the new total from the accumulator

So far, you have been taught to add several COSTs into TEMP by writing CLA COST, ADD TEMP, STR TEMP. Of course, the program will work just as well if you switch the first two addresses, making it CLA TEMP, ADD COST, STR TEMP. We have been doing it the first way for a reason you will appreciate in the next section.

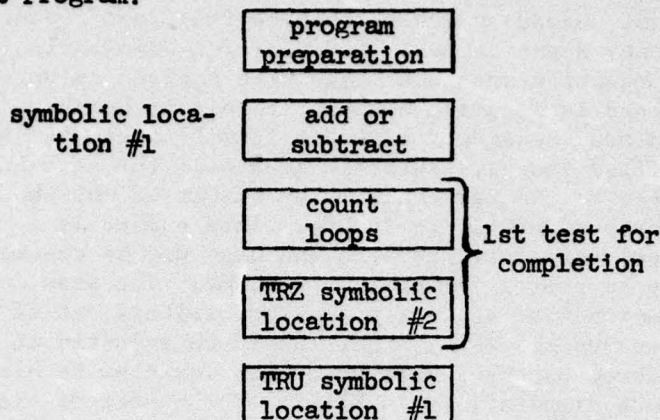
However, if you wanted to subtract COST from TEMP many times, you would have to reverse the addresses, making the order similar to the counting instructions in the test for completion:

CLA TEMP
SUB COST
STR TEMP

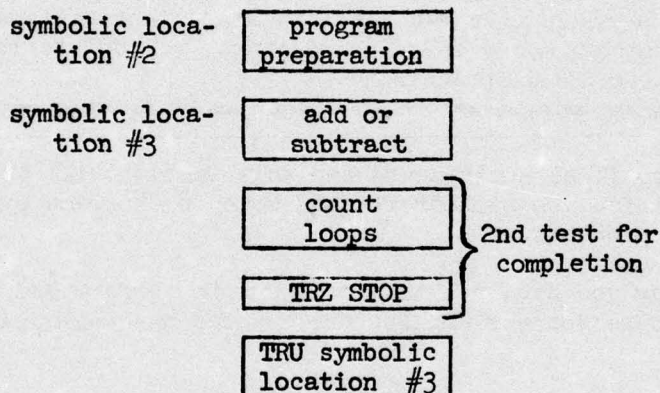
There is no reason why programs must be restricted to just one set of looping instructions. You may want to loop for one kind of addition or subtraction, and then transfer down to start a second set of loops. In effect, you will have two separate programs written end to end.

Only one modification is necessary to handle this type of problem, namely, the TRZ instruction in the first test for completion should transfer to a symbolic location starting the second program rather than to STOP:

1st Program:



2nd Program:



STOP HLT

Problem 6.4. A supply sergeant has received a request for a large order of two types of vacuum tubes. He has neither type in stock and needs a program to compute the cost of ordering them from a supply house. The cost of each type of tube is located relative to TUBE. The number of each type requested is located relative to ASK. Assume that both numbers are greater than zero. Temporary storage for these two numbers is available relative to COUNT. The total cost of all tubes should be stored in TOTAL. Symbolic locations to be used are: FIRST—to complete the loop in the first program; SECOND—to start the second program; and THIRD—for looping in the second program. Location CON contains the number 1.

Problem 6.5. The total funds available to a mess sergeant depend on money received from Post Finance and money spent on meals for visiting dignitaries. The sergeant starts the month with a given amount, contained in location AVAIL. To this number he should add the amount received from Finance, which is a fixed amount, contained in MEAL1, for each man assigned to his mess hall. The number of men who eat there regularly is in MEN. This number is obviously greater than zero and need not be checked. It can be stored temporarily in TEMP. The mess sergeant has to pay for meals for visitors out of his own funds. Thus, AVAIL should be adjusted to take these expenses into account. The cost to him for each dignitary is in MEAL2. The number of visitors last month is in VIP. There may not have been any visitors, so this number should be checked for zero. It can be stored in VISIT. Location K contains a 1. Use IN as the symbolic location for looping to compute funds received, NEXT to start instructions subtracting money spent on visitors, and OUT to loop for repeated subtractions.

Two final refinements and this section will be complete. We will treat them both in the same practice problems.

Say you have two sets of loops to perform and you're not sure whether either loop counter has anything in it.

As part of program preparation for the first looping program, you would clear-and-add the first loop counter and TRZ to the symbolic location for the second program, rather than STOP. In other words, if there are zero items of the first type, you can skip the entire first program and start the second. If the second loop counter also has nothing, you can TRZ to STOP as usual, or to final instructions if there are any.

Finally, you will often need to combine the results of both programs by adding the two answers together or subtracting one from the other. This you can do in a third little program at the very end. This requires that the symbolic location used with TRZ in the second test for completion should refer to the final instructions, rather than STOP.

These kinds of problems block out as follows:

1st Program:

CLA 1st counter
TRZ symbolic lo- cation #2
STR temporary location
Clean garbage from both locations

1st program
preparation

symbolic location 1 Add or subtract

Count loops
TRZ symbolic lo- cation #2

1st test for
completion

TRU symbolic lo-
cation #1

2nd Program:

CLA 2nd counter
TRZ symbolic lo- cation #4
STR temporary location

2nd program
preparation

symbolic location 3 Add or subtract

Count loops
TRZ symbolic lo- cation #4

2nd test for
completion

TRU symbolic lo-
cation #3

3rd Program:

symbolic location 4 Add or subtract
 answers
 HLT

AD-A034 045

HUMAN RESOURCES RESEARCH ORGANIZATION ALEXANDRIA VA
BASIC COMPUTER PROGRAMMING: A SELF-INSTRUCTIONAL COURSE. (U)
JUN 67 R J SEIDEL, H G HUNTER, I C ROTBERG

F/G 9/2

DA-44-188-ARO-2

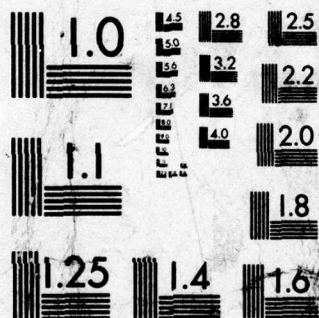
UNCLASSIFIED

NL

2 OF 2

AD
A034045





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Problem 6.6. Write a program to compute the total value of two kinds of typewriter in stock, separately, and the grand total. Sales have been good, and one or both may be sold out. Addresses to be used are:

	<u>First Kind</u>	<u>Second Kind</u>
Price	PRICE1	PRICE2
Number left	NUM1	NUM2
Storage	COUNT1	COUNT2
Answers	TOTAL1	TOTAL2

Store the grand total in BIGTOT. Use TYPE1 for looping in the first program, NEXT to start the second program, TYPE2 for looping in the second program, and BOTH for final instructions. Location DIGIT contains the number 1. Hint: we can zero out both answer locations with one block of instructions.

Incidentally, a little program inside a bigger one is often called a "subroutine."

Problem 6.7. Write a program to compute:

$$\text{ANSWER} = (\text{COST} \times \text{ITEMS}) - (\text{A} \times \text{B})$$

Don't be frightened. Multiplying one number by another, such as COST x ITEMS, is exactly the same thing as adding one number as many times as another. You have done this by using one number in the adding instructions and the other as the loop counter. For example, 12 x 3 can be obtained by adding 12 three times, using a 3 in the loop counter. In short, multiplication is simply a form of repeated addition. Set up your program by getting COST x ITEMS in one routine, or set of instructions, using symbolic location LOOP for looping, ITEMS as the loop counter, and FIRST for the answer. Start the second routine to get A x B with symbolic location NEXT, using B as the loop counter, LOOP2 for looping, and SECOND for the answer. Subtract one answer from the other at symbolic location LAST, putting the result in ANSWER. Do not assume that any of the locations--COST, ITEMS, A, or B--are greater than zero, but do not bother to save any of these numbers either. (If either COST or ITEMS contains a zero, COST x ITEMS = 0, you can skip to the routine computing A x B). Location KON contains the number 1.

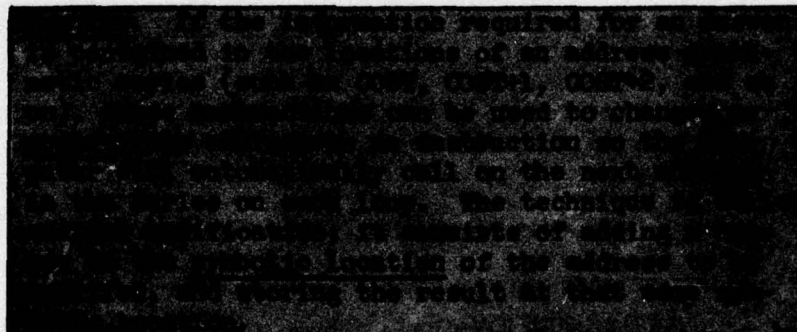
PHASE III

DATA PROCESSING

This section talks about ways to get information from many different locations with just a few instructions. So far, you have had to address each location individually. Now you will learn how to tell the computer to call on an entire series of locations, using a technique called address modification, which is explained in Part One.

You will also learn ways to sort out the locations visited into various types, how to count the number of addresses of each type, add their contents, relocate one series into another, or some combination of the preceding. All of these techniques do something with, or process, the information, or data, necessary for an answer. Thus, these techniques are all related to data processing. They are discussed in Parts Two through Four. Part Five is devoted to review and practice.

PART ONE: ADDRESS MODIFICATION



Say you have an address arithmetic series that starts off this way:

$\frac{\text{COST}}{4}$	$\frac{\text{COST}+1}{10}$	$\frac{\text{COST}+2}{2}$	$\frac{\text{COST}+3}{8}$	and so on
-------------------------	----------------------------	---------------------------	---------------------------	-----------

Notice that each location contains a different number.

Now let's say you want to add the contents of the entire series, placing the total in ALL.

The basic looping program you have used before (shown below without a test for completion) will do it if we can change the address COST before each new loop, modifying it to COST+1 before the second loop, to COST+2 before the start of the third loop, and so on.

```
          CLA ZRO
          STR ALL
OVER      CLA COST
          ADD ALL
          STR ALL
          TRU OVER
```

If we can modify that one address each time, the program will add in COST, then COST+1, then COST+2, and so on, through the entire address arithmetic series.

COST is part of an instruction that had OVER as its symbolic location. That is, COST appears in the instruction OVER CLA COST. COST is the address to be modified and OVER is its symbolic location.

An address can be modified by working with its symbolic location.

For example, adding a 1 to a symbolic location adds a 1 to the address appearing with that symbolic location.

Consider OVER CLA COST. The symbolic location is ____.

Changing the symbolic location OVER changes the address ____.

Adding a 1 to OVER serves to add a 1 to ____, changing it to COST+1.

In LOWER STR CASE, the address CASE can be changed to CASE+2 by adding a ____ to LOWER.

In AFTER SUB COATS, COATS can be modified to COATS+1 by ____ a ____ to ____.

Numbers can be added to symbolic locations in the same way you have added to addresses. Simply write the symbolic location in the address field. For example, the following instructions add a 1 to the symbolic location OVER.

```
CLA OVER
ADD ONE
STR OVER
```

Now let's go back to the basic adding program, this time including a new block of three instructions to modify COST.

```
          CLA ZRO      cleaning out
          STR ALL      garbage

OVER      CLA COST
          ADD ALL      adding
          STR ALL

          CLA OVER     address
          ADD ONE      modification
          STR OVER

          TRU OVER     looping
```

The first adding instructions add the contents of COST into ALL, as you know. The next three instructions, for address modification, add a 1 to COST by adding a 1 to its symbolic location OVER, changing COST to COST+1.

As a result, when the computer starts the second pass, it will CLA COST+1 rather than COST. Address modification has changed the address in the instruction CLA COST, modifying it to COST+1.

OVER

COST

Address modification is accomplished with a block of three instructions that change the address by adding to its symbolic location.

COST

2

Problem 1.1. You have just written REPEAT ADD PRICE. Write the three address modification instructions that will change PRICE to PRICE+1. There is a 1 in location ONE.

adding, 1,
AFTER

Problem 1.2. HUBERT SUB MEN appears earlier in your program. Write the instructions that will modify MEN to MEN+1, if there is a 1 in DIGIT.

Problem 1.3. You have instructed the computer to LOWER STR CHECK earlier in your program, and you want it to perform STR CHECK+2 on the next loop. Write the instructions you would insert to accomplish this. There is a 1 in ONE.

Problem 1.4. Write a program to add the contents of the entire series of locations addressed relative to VALUE, storing the total in FINAL. Use REPEAT for looping. There is a 1 in location K.

The solution requires you to zero out the answer location, as usual, and write two blocks of instructions, one for adding, which you have done before, and a second block for address modification. Don't worry about the test for completion; we will insert it later.

Problem 1.5. Write a program to add the contents of every other location in the series relative to COST (COST, COST+2, COST+4, etc., skipping COST+1, COST+3, etc.). Store the answer in EVERY. The number 2 is in location TWO. Use AGAIN for looping.

Address modification changes addresses in an address arithmetic series by working with a symbolic location. It has nothing to do with commands.

Here's a program adding the contents of the entire address arithmetic series relative to BLOCK, placing the answer in HOUSE.

	CLA ZRO	cleaning out garbage from
	STR HOUSE	the answer location
SYMBOL	CLA BLOCK	
	ADD HOUSE	BLOCK is put into HOUSE
	STR HOUSE	
	CLA SYMBOL	BLOCK is changed to BLOCK+1
	ADD ONE	by adding a 1 to its
	STR SYMBOL	symbolic location
	TRU SYMBOL	

On the second pass, BLOCK+1 is added into HOUSE. Then BLOCK+1 is changed to BLOCK+2 during address modification. Consequently, the third pass adds in BLOCK+2. BLOCK+2 is modified to BLOCK+3 during the remainder of the pass, and BLOCK+3 is added in at the start of the next pass, and so on.

Up to now the loop counter in the test for completion has referred to the number of repeated additions (or subtractions) using the same address.

With address modification, however, the address is different on each loop. After the first loop, you are no longer going back to add in COST, for example, but COST+1, then COST+2, and so on, adding in a different location in the series each time.

Therefore, the loop counter in a program with address modification refers to the number of different locations to be called on.

In this regard, remember that the number tacked on behind an address in an address arithmetic series is one less than its position in that series. For example, COST+2 is not the second address in the series, but the third.

If you wanted to add in the contents of locations PRICE through PRICE+3, the loop counter should contain the number ____.

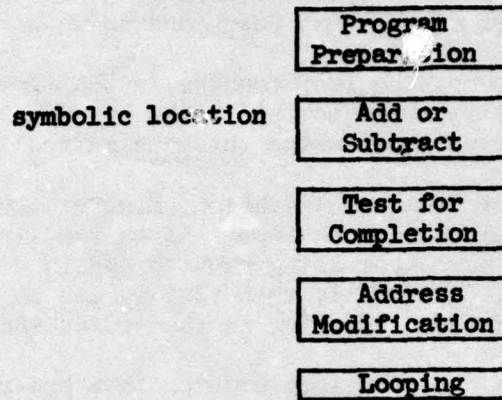
To add the contents of every other location in the series COST through COST+4, the loop counter should have the number ____.

If you didn't understand the last question, look at the complete series:

COST
COST+1
COST+2
COST+3
COST+4

Every other location would take in COST, COST+2, and COST+4, for a total of three locations, requiring three loops and therefore a 3 in the loop counter.

The test for completion can be inserted right after adding, as you have always done, followed by address modification, like this:



STOP HLT

Here's an example to demonstrate how a complete program might look. It adds TIE through TIE+9 into ANSWER.

	CLA TEN	Put a 10 into the loop
	STR COUNT	counter
	CLA ZRO	Zero out the answer
	STR ANSWER	location
CRAVAT	CLA TIE	Add in the first address
	ADD ANSWER	of the series
	STR ANSWER	
	CLA COUNT	
	SUB ONE	Subtract a 1 for the ad-
	STR COUNT	dition just completed
	TRZ STOP	
	CLA CRAVAT	
	ADD ONE	Modify TIE to TIE+1
	STR CRAVAT	
TUBE+4	TRU CRAVAT	Loop back for TIE+1
	STOP HLT	
		TUBE+1

Problem 1.6. A soap company records the sales made by each salesman, stored relative to SALES. That is, the number of sales made by the first salesman is in SALES, the number made by the second is in SALES+1, and so on. Write a program to compute the total sales of all salesmen, placing the answer in TOTAL. The company employs 16 salesmen. There is a 16 in MEN, which can be saved by storing it temporarily in TEMP. There is a 1 in location ONE. Use SELL for looping (and address modification).

Problem 1.7. A large music store sells records of many different types, such as classical, western, folk, and so on. The number of different types is in TYPES. The sales of each type record is stored in sequence relative to RECORD. Write a program to compute the total number of records sold, of all types, placing the answer in SALES. Use POP as storage for the number of types (for the loop counter), and HIT for looping. There is a 1 in location K1.

Often several different memory locations, or words, are reserved for information about just one item. For example, a warehouse might keep three kinds of information on each tube; its size, its cost, and its present location.

Each tube is said to have a "three-word record." This means three consecutive locations in an address arithmetic series are set aside for each tube.

For example, assume the series relative to TUBE is used for storing 3-word records on each tube, their size, cost, and location:

<u>First Tube</u>			<u>Second Tube</u>		
<u>Size</u>	<u>Cost</u>	<u>Location</u>	<u>Size</u>	<u>Cost</u>	<u>Location</u>
TUBE	TUBE+1	TUBE+2	TUBE+3	TUBE+4	TUBE+5

The cost of the first tube is in TUBE+1. The cost of the second tube is in ____.

If you were going through the series to pick up the total cost of all tubes, you would start with ____.

To get from TUBE+1 to TUBE+4, you would need to add the number to the appropriate symbolic location during address modification.

That is, you want to add in the contents of every third location, starting with TUBE+1. This means a constant of 3 must be added to the symbolic location during address modification.

Try another example. A publishing company maintains a 3-word record on each of its books, stored relative to BOOK. The words are date of publication, retail price, and number of sales. What location contains the number of sales of the first book in the series?

Where would you go to get the number of sales of the second book?

What number should be added during address modification to call on the locations containing the number of sales of all books?

two

Problem 1.8. Each vacuum tube in stock has information in its record in the following order of words: brand name, date last used, place last used, and hours used. These words are stored relative to TUBE. The total number of tubes in stock is in ITEMS. Use HOLD as temporary storage. Write a program to compute the total number of hours all tubes have been used and store this value in HOURS. There is a constant of 1 in ONE and a 4 in FOUR. Use REPEAT for looping.

Problem 1.9. The pay record for each man in a business firm has information in the following order of words: badge number, shift worked, regular pay, and overtime pay. These words are stored relative to RECORD. The total number of employees is in HELP. Compute the total regular pay for all employees, placing the amount in REGPAY. There is a 1 in CON1 and a 4 in CON4. Use TEMP for temporary storage and TOTAL for looping.

3

PART TWO: SORT-AND-COUNT PROBLEMS

BOOK+2

3

BOOK+5

Shown below are the first five locations in the series relative to MAN. Each location represents a different driver and contains the number of accidents that man has had. The first man has had none, the second has had 4, and so on.

MAN	MAN+1	MAN+2	MAN+3	MAN+4
0	4	0	0	2

The Registry of Motor Vehicles wants to know how many drivers in this series have had at least one accident. How many have?

Since we can't look inside the computer to count the offenders ourselves, we need a program to get the information out.

We can count the numbers greater than zero by copying the contents of each location into the accumulator and asking, "Is this number a zero?" But if the answer is no, the number must be greater than zero, and we can count it by adding a 1 to the counter set aside for traffic offenders.

What instruction will copy the contents of the first location into the accumulator?

What command asks whether the number in the accumulator is a zero?

Now let's back up for a moment to fill in the rest of the program before returning to the sort-and-count portion. There are five locations to be processed, so we need a test for completion that has the number in its loop counter. Assume COUNT has that number.

Then we will need some address modification instructions to tell the computer to visit a different location on each pass through the loop. The complete program, minus sort-and-count instructions (and program preparation) is shown below.

REPEAT	CLA MAN	
	sort-and count instructions	
TEST	CLA COUNT	
	SUB ONE	the test for
	STR COUNT	completion
	TRZ STOP	
	CLA REPEAT	address
	ADD ONE	modification
	STR REPEAT	
	TRU REPEAT	
STOP	HLT	

If the number in MAN is a zero, we can forget about it and skip directly to the test for completion to finish out the loop. What instruction will transfer the program to the test for completion if MAN has a zero?

If MAN does not have a zero, the driver must have had at least one accident. Will the program transfer if MAN does not have a zero?

Assume you are counting accident violators in location BANG. What instructions will add a 1 to that location?

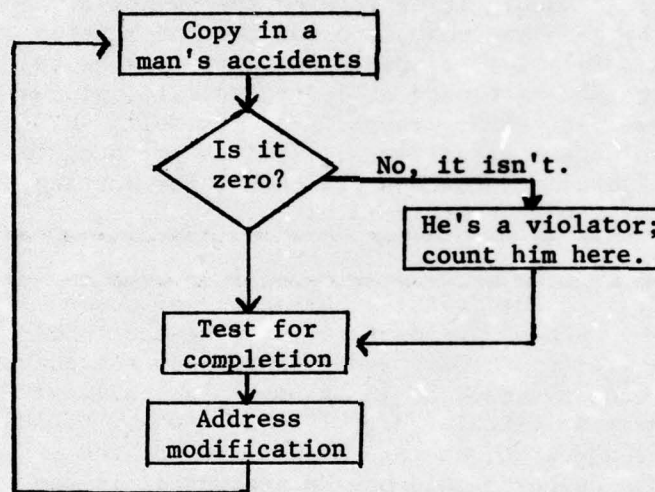
In short, the data processing instructions for this sort-and-count problem are:

CLA MAN

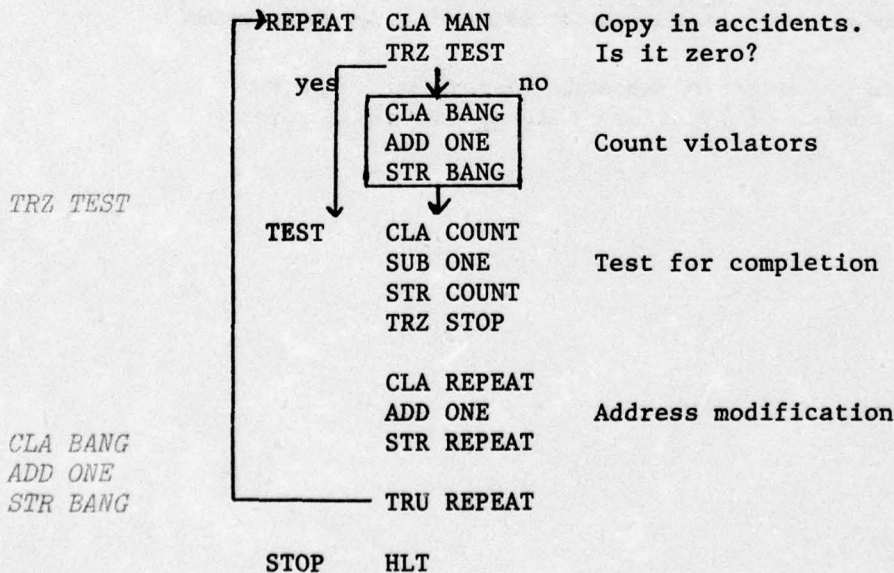
TRZ

CLA MAN Copy in the number from MAN.
 TRZ TEST If it's a zero, skip to the test for comple-
 CLA BANG tion. If it's not a zero, it must be greater,
 ADD ONE identifying an accident violator. Count him
 STR BANG by adding a 1 to BANG.

Look at it in terms of a block diagram. You are counting offenders in location BANG. But you want to add a 1 to that location only if the driver being checked has not had zero accidents. If he has had any at all, the number will not be zero.



Compare the diagram with the complete program:



No

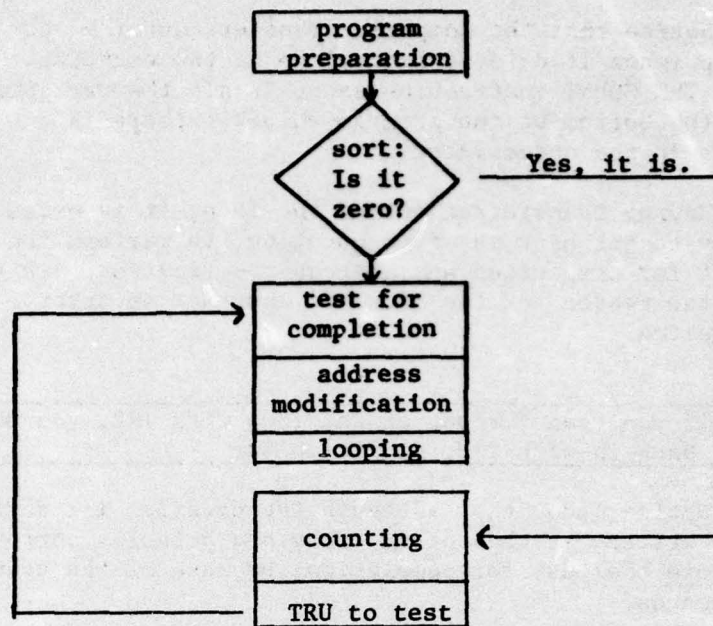
To count the number of locations that do not contain zeroes, copy in the contents of each location and TRZ to the test for completion, counting the locations that are not eliminated right after the TRZ instruction:

1. Clear-and-add a number to be checked.
2. TRZ to the test for completion.
3. Add a 1 to a counter; these instructions will be performed only if the computer has not skipped past them on the TRZ instruction.

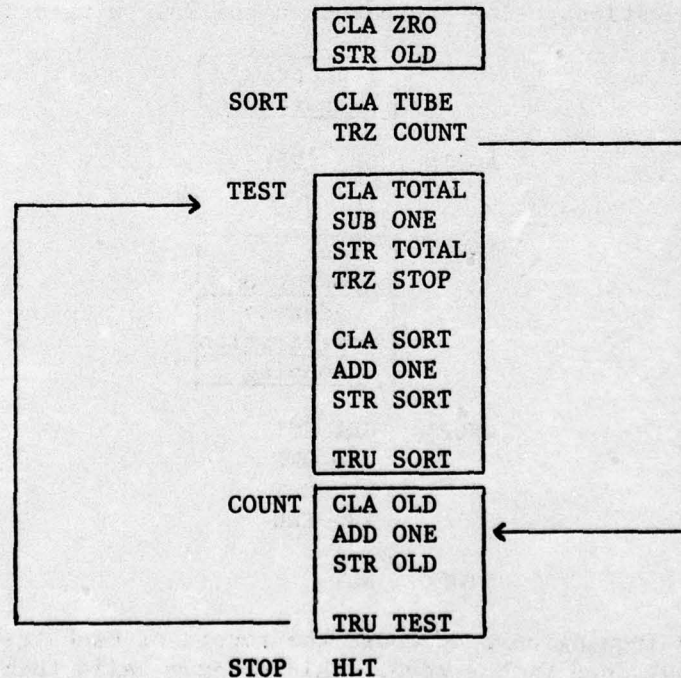
Problem 2.1. A manufacturer records the number of sales made by each salesman, storing the information in the series relative to PUSH. He wants to know the number of men who have made at least one sale, giving him the answer in BONUS. There is a 1 in UNIT. MEN contains the number of salesmen. It does not need to be saved. Symbolic locations are CHECK, for sorting, and TEST, for the test for completion.

Problem 2.2. A clinic records the number of house calls made by each of its doctors, storing the information relative to DOCTOR. It wants to know the number of doctors who have made at least one house call, giving the answer in CALLS. There is a 1 in ONE. MEDICS contains the number of doctors working out of the clinic. This number should not be destroyed; it can be saved in TEMP. Symbolic locations to be used are AGAIN, for looping, and OUT, for the test for completion. Assume at least one doctor.

Now turn the question around. Assume you want to count the number of locations that do contain a zero. Here's the diagram:



For example: A supply sergeant codes new tubes with a 1 and old tubes with a 0, storing the information relative to TUBE. The number of tubes on hand is in TOTAL. The program below counts the number of old tubes, placing the answer in OLD. Compare it with the diagram.



CLA CUT

Notice that the computer transfers outside the loop (when it detects a zero) to do the counting. The TRZ COUNT instruction sends it all the way down to the bottom of the program whenever there is a zero in the accumulator.

no,
yes

Having transferred out of the loop, it is necessary to get back in after counting, to perform the test for completion and address modification. This is the reason for the TRU TEST instruction after counting.

after

If you transfer out of the loop with TRZ, you must get back in with TRU, to finish it.

all trees

Notice, too, that although the counting instructions are written at the bottom, they are actually performed before the test for completion, because of the transfer commands.

All locations must be processed, so all loops must end with a test for completion and address modification. The TRZ instruction interrupts the loop momentarily to increase a counter whenever a loop starts with zero in the accumulator.

Questions below are based on the following program:

	program preparation
AGAIN	CLA TREE TRZ CHOP
END	test for completion address modification looping
CHOP	CLA CUT ADD ONE STR CUT TRU END
STOP	HLT

A logging company codes the record of each tree to be cut down with a zero. This program tells them how many trees are marked for cutting.

Assume TREE has a zero. What instruction is performed after TRZ CHOP?

If TREE has a zero, is the test for completion performed before or after the counting instructions starting at symbolic location CHOP?

If TREE does not have a zero, will it be counted in location CUT? Will a test for completion be performed?

Do you want to run a test for completion and address modification on all trees, or just those which are to be counted?

To count locations with zeroes, copy the contents of each location into the accumulator, TRZ to the end of the program for counting, and TRU after counting back to the test for completion.

Problem 2.3. A branch office of the Internal Revenue records whether each citizen has paid his taxes by coding a 1 in the series relative to TAXES if he has paid, and a 0 if he has not. Complete the program below to count the number of persons who have not yet paid their taxes, storing the count in NOTYET. There is a 1 in K1.

program
preparation

REPEAT _____

TEST test for
 completion

address
modification

GETHIM _____

STOP HLT

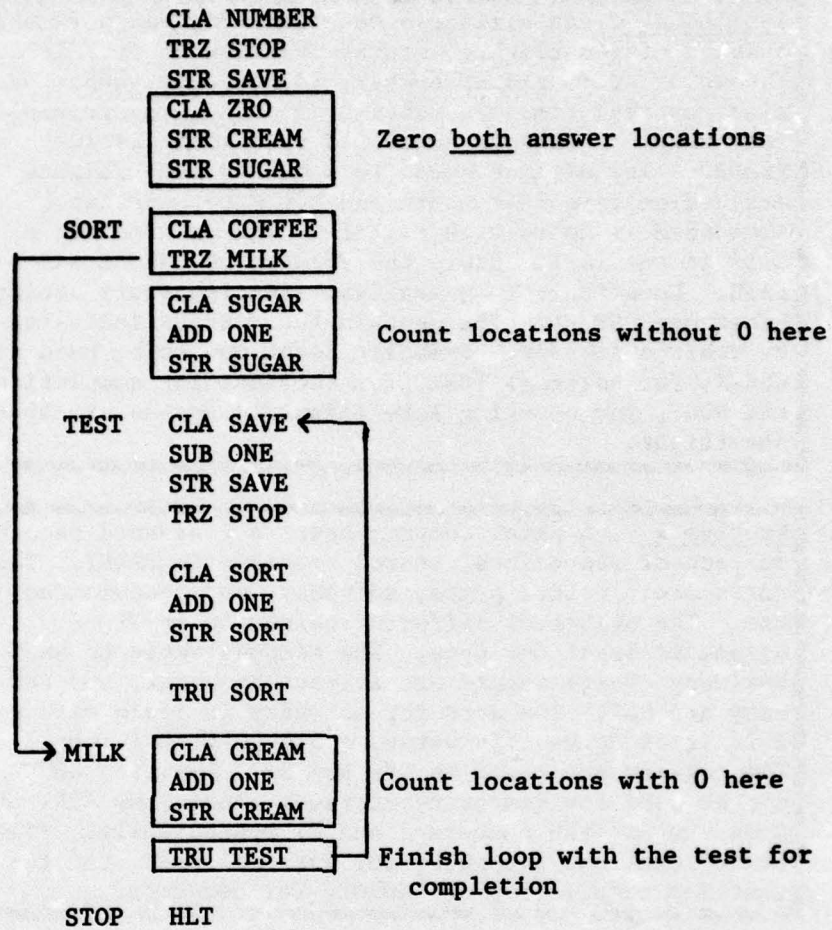
Problem 2.4. An Army battalion maintains a four-word record on each man, stored relative to NUMBER. The words are: serial number, unit, rank, and MOS. PERSON contains the number of men on record. Use FILE to store the number of men temporarily. Rank is indicated by a 1 for officer and a 0 for enlisted personnel. The battalion wants a program to determine the number of enlisted personnel, storing the answer in EM. Location K1 contains a 1 and K4 has a 4. Symbolic locations to be used are REPEAT, for sorting; FINAL, for the test for completion; and ENLIST, for counting the number of enlisted personnel.

Problem 2.5. A business firm maintains a six-word record on each employee, relative to WORKER. They are: assigned branch, specialty, salary, marital status, number of children, and years of service. FIRM contains the number of personnel in the firm. Use SAVE to store the number of employees temporarily. Marital status is indicated by a 1 for married and a 0 for single. The firm wants a program to determine the number of single employees. The information can be stored in SINGLE. Location ONE contains a 1 and location SIX has a 6. Symbolic locations are FREE, for counting; CYCLE, for looping; and DONE, for the test for completion.

Counting both kinds of locations, those containing a zero and those that do not, requires both sets of counting instructions, the block right after sorting (for locations without zeroes) and the block at the end (for locations with zeroes).

Nothing new. Just a longer program. But don't forget to zero out both answer locations as part of program preparation.

The following example checks the series relative to COFFEE, counting locations with zero in CREAM at the bottom, and locations without zero in SUGAR right after sorting. Both kinds of counting are followed by the test for completion. The number of locations in the series is in NUMBER (the loop counter), which is stored temporarily in SAVE.



Problem 2.6. An airline maintains a five-word record on each of its flights, stored relative to FLIGHT. The words are: flight number, flight date, departure time, arrival time, and whether it was late arriving. The number of flights scheduled each month is in PLANES. The airline needs to know how many flights arrived on time last month and how many were late. That word is coded with a 1 if it was on time and a 0 if it was late. Store the answers in ONTIME and LATE. Location FLY is available for temporary storage. Locations ONE and FIVE contain the numbers indicated by their addresses. Symbolic locations to be used are CHECK, for sorting; TEST, for the test for completion; and GOOF, for counting late flights. Assume at least one flight.

Problem 2.7. A paint company keeps a four-word record on each of its paints, stored relative to PAINT. The words are: color, price, solvency, and recommended use. The number of different paints is in TYPES. Assume at least one type. The company wants to know how many of its paints are solvent in water, and how many are not. The word for solvency is coded with a 1 if it is solvent in water, and a 0 if it is not. The answers are to go in YES and NO. Location COUNT can be used for temporary storage. Locations CON1 and CON4 contain the numbers 1 and 4, respectively. Symbolic locations are LOOP, for sorting; LAST, for the test for completion; and NOSOL, for counting.

PART THREE: SORTING TECHNIQUES

Example. So far you have been asking whether a list contains a word, sorting out those that do from those that don't with the GO command. Often, however, the question you want answered is not whether a location contains a word, but how many other words it contains. These problems can be solved with your new conditional instruction, counting the number of 1's in a word. For example, if you have a list of words, you can count the number of 1's in each word, and then sort the words by the number of 1's. This is a useful technique for many applications.



Sorting by Subtraction

Say you have a problem asking how many locations contain the number 5 and how many do not. The solution is simple. Bring the contents of each location to the accumulator and subtract a 5. If the number was a 5, it is now a zero, and can be identified using TRZ, as before.

Thus, the solution involves only one additional instruction, inserted into the block for sorting:

1. CLA the address
2. SUB a constant equal to the number you want to identify
3. TRZ to the counting instructions

If you want to count the number of locations containing the number 8, what would you subtract before using a TRZ instruction?

Assume COST contains an 8, and you want to count 8's. CLA COST copies the 8 into the accumulator, and SUB EIGHT changes the number there to a zero. (The number in COST is not changed; it is still 8.) TRZ can then transfer the program to the counting instructions.

In short, you can identify a location with an 8 by copying its contents into the accumulator, subtracting 8, and using TRZ.

A supply sergeant maintains information on each pair of pants in stock, including waist size. Pants with a 28-inch waist are coded with a 28, 30-inch pants are coded 30, and so on. What instruction should you insert before using TRZ to identify all the 32-inch pants, if locations K28, K30, and K32 contain the numbers 28, 30, and 32?

The problem asks for the number of locations containing a 32. The solution is to copy the contents of each location into the accumulator and subtract 32. If the location has a 28, the number in the accumulator after subtraction will be -4 and TRZ will not pick it up. If it is 30, TRZ still will not detect it. But if it is 32 in the location, it will be zero in the accumulator, and TRZ will transfer the program to the counting instructions.

STR THERE

In general, to identify a particular number, copy the contents of each location into the accumulator and subtract the number you want to pick out. Then use TRZ as usual.

Problem 3.1. A bra manufacturer records the size of each bra in stock in the series relative to BRA. A-cup bras are coded with a 1, B-cup with a 2, C with a 3, and D with a 4. The total number of bras in stock is in TOTAL. The manufacturer wants to know how many B-cup bras he has on hand. Write a program to compute the answer and place it in MEDIUM. NUMBER is available for storage. ONE and TWO contain the numbers 1 and 2. Use REPEAT for looping, LAST for the test for completion, and COUNT for counting.

STR THAT
STR THAT

Problem 3.2. A company maintains a five-word record on each employee, stored relative to WORKER. The words are: salary, position, years employed, marital status, and social security number. The company hands out service watches after 20 years of employment. Write a program to count the number of employees who are eligible for their watch this year, storing the answer in LOYAL. The total number of employees is in TOTAL, which can be saved by storing it temporarily in TEMP. Locations ONE, K5, and K20 contain the numbers 1, 5, and 20, respectively. Use CHECK for sorting, OVER for the test for completion, and TWENTY for counting. Assume at least one employee.

SUB K3
TRN COUNT

Sorting with TRN

-1, -2,
-3

Numbers less than zero are called negative numbers, as you may know. For example, 6 minus 8 results in a negative number, minus two (-2).

The computer can sense a negative number with the TRN command, which stands for "transfer if negative."

Thus, if the number in the accumulator is less than zero, the program will transfer on TRN.

What instruction will be performed after TRN COUNT below, if THIS contains a 5 and SIX has a 6?

```
CLA THIS
SUB SIX
TRN COUNT
STR THAT
COUNT STR THERE
```

$THIS - SIX = 5 - 6 = -1$, which is a negative number.

What will be performed above if THIS starts with a 6? With a 7?

TRN will cause a transfer only if the number in the accumulator is negative. If it is zero or positive, the program will not transfer but will move directly to the next instruction.

The TRN command can be used in sorting to pick up any number less than zero. The TRZ command is limited to only one number, the number zero, while TRN can detect an entire range of numbers.

TRN with a sorting problem usually requires subtraction first.

A dental office records the number of visits made by each patient, stored relative to TOOTH. What two instructions should you write after CLA TOOTH to pick out patients making less than three visits, counting them at symbolic location COUNT, if locations K1, K2, and K3 contain the numbers 1, 2, and 3, respectively?

In the previous example, you were interested in counting 2's, 1's, and 0's, the numbers less than 3. If the number was originally a 2, subtracting 3 makes it the number _____, subtracting 3 from 1 gives _____, and 3 from 0 is _____, all negative numbers.

What number should be subtracted to pick up just the 1's and 0's?

2

TRN, TRZ

7

Problem 3.3. The First Sergeant has decided to assign weekend passes on the basis of the last inspection. All enlisted personnel have a five-word record stored relative to EM. The words are: squad number, rank, years of service, number of gigs (deficiencies) on the last inspection, and date of rank. The total number of enlisted personnel is in LIST. Write a program to determine how many men got less than 3 gigs on the last inspection, storing the result in PASSES. Locations ONE, TWO, THREE, FOUR, and FIVE contain the numbers suggested by their addresses. Use MEN as temporary storage for the loop counter, CYCLE for looping, END for the test for completion, and DETAIL for counting. Assume at least one man.

Problem 3.4. The Division Training Center maintains a four-word record on each soldier who went on maneuvers, stored relative to TRAIN. The words are: numbers of aggressors he "killed," his squad leader's rating, number of times he got lost, and his score on debriefing. The total number of men who went on maneuvers is in FIGHT, which can be stored temporarily in FIGURE. Write a program to count the number of men who got lost more than twice, storing the answer in LOST. There is a constant of 1 in CON1, a 2 in CON2, a 3 in CON3, and a 4 in CON4. Symbolic locations are MORE, for looping, and DONE, for the test for completion. (Counting occurs right after sorting.)

Multiple Sorting

TRN by itself can sort locations into those containing negative numbers and those that do not. TRZ by itself can sort on the basis of zero and not-zero. Together, they can divide the range of numbers into three categories: negative (TRN), zero (TRZ), and positive (whatever is left).

In brief: CLA address

TRN to count negative numbers

TRZ to count zeroes

Count positive numbers here, after
negative and zero numbers have been
eliminated

Usually, some number must be subtracted before TRN and TRZ can be used. When sorting into three categories, subtract a number that will make the "middle" number a zero.

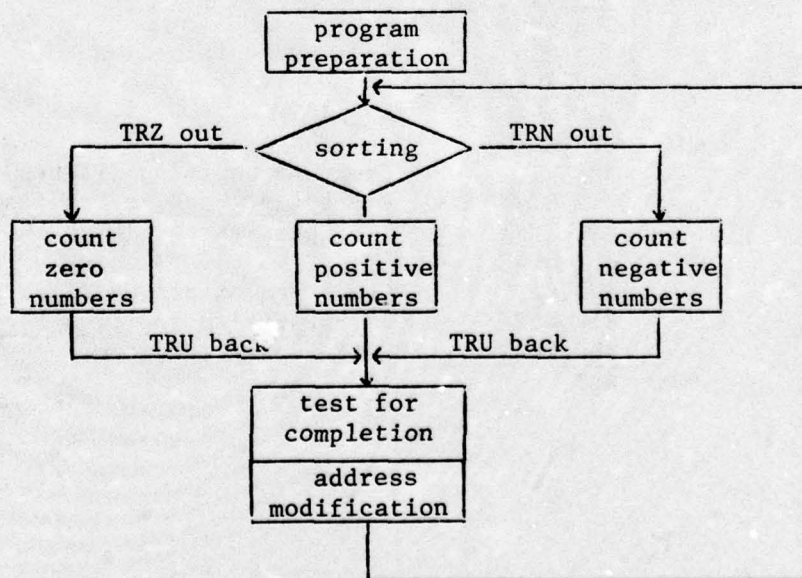
For example, in sorting for 1's, 2's, and 3's, first subtract the number _____, to change them to -1, 0, and +1.

In sorting for 3's, 7's, and 8's, first subtract the number _____.

Then TR__ to count the 3's, and TR__ to count the 7's.

Of course, if there are three answer locations, all three must be zeroed out as part of program preparation and any time you transfer out of the loop to count, you must get back in to run the test for completion and address modification.

Previously, you have transferred out to count one kind of number. Now you will transfer out to count two types. Just remember that each time you go outside the loop, you need a TRU instruction that will return you to the test for completion.



Here's an example of a complete program. It's long, but still manageable if you think in terms of blocks.

An airline company codes the time of day each flight leaves, with a 1 for morning, 2 for afternoon, and 3 for evening. The information is stored relative to FLIGHT. The number of flights is in FLY, which can be saved by storing it in TEMP. (There are obviously more than zero flights.) Write a program to count the number of morning, afternoon, and evening flights, placing the answers in MORN, NOON, and NIGHT. Use WHEN for looping, TEST for the test for completion, AM to count morning flights, and PM to count afternoon flights. Locations K1 and K2 contain a 1 and 2.

	CLA FLY	Save the loop counter without
	STR TEMP	checking
	CLA ZRO	
	STR MORN	Zero out all three answer
	STR NOON	locations
	STR NIGHT	
WHEN	CLA FLIGHT	
	SUB K2	
	TRN AM	1-2= -1; these are morning flights
	TRZ PM	2-2= 0; these are afternoon flights
	CLA NIGHT	
	ADD K1	Anything left started off as a 3,
	STR NIGHT	indicating an evening flight
TEST	CLA TEMP	
	SUB K1	Test for completion
	STR TEMP	
	TRZ STOP	
	CLA WHEN	
	ADD K1	Address modification
	STR WHEN	
	TRU WHEN	Looping
AM	CLA MORN	
	ADD K1	Count the morning flights and
	STR MORN	<u>transfer back to the test for</u>
	TRU TEST	<u>completion to finish the loop</u>
PM	CLA NOON	
	ADD K1	Count the afternoon flights and
	STR NOON	<u>transfer to the test for completion</u>
	TRU TEST	<u>to finish the loop</u>
STOP	HLT	

Problem 3.5. An electronics warehouse wants a program to count the number of tubes used in January and the number in February, placing the answers in JAN and FEB. The words for each tube are in memory starting at location VACUUM. The words are: part number, date received, shipment number, manufacturer, number used, and place used. The month is coded by number, with a 1 for January, a 2 for February, and so on. Locations ONE, TWO, and SIX contain the numbers 1, 2, and 6, respectively. Use HOLD as temporary storage for TUBES, which contains the number of tubes in the warehouse. Symbolic locations are COMPUT, for sorting; LAST, for the test for completion; MONTH1, to count January tubes; and MONTH2, to count February tubes.

Problem 3.6. A shirt manufacturer stores information on the size of each shirt relative to SHIRT, with a 2 for small, a 4 for medium, and a 6 for large. Write a program to count the number of small and large sized shirts, placing the answers in SMALL and LARGE. The total number of shirts is in TOTAL, which can be stored in NUMBER. Location KON1 has a 1, and KON4 has a 4. Use symbolic location SORT for sorting, END for the test for completion, and COUNT for counting.

TRN picks out any number that is negative, while TRZ picks out only one number, a zero. Thus, if you want to count specific numbers, it is easiest to use TRZ. For example, to pick out 3's, 7's, and 8's in the series relative to COST:

	Original Number		
CLA COST	3	7	8
SUB THREE	-3	-3	-3
TRZ to count 3's	0	4	5
SUB FOUR		-4	-4
TRZ to count 7's		0	1
SUB ONE			-1
TRZ to count 8's			0

In other words, subtract the number you want to identify from itself, making it zero so you can TRZ to pick it out and transfer to count it.



CLA HATS
STR GLOVES

Problem 3.7. Officer candidates are given a test with a maximum of 10 points. Scores from 0 to 7 are flunking, an 8 is provisional acceptance, and a 9 or 10 means unqualified acceptance. The testing center maintains a four-word record on each candidate—serial number, test score, age, and parent organization—stored relative to OFFCAN. Write a program counting the number of students scoring in each category, storing the counts in FLUNK, MARGIN, and ACCEPT. The number of candidates is in PUPIL. This number is greater than zero and can be stored in MEN. Locations ONE and FOUR contain the numbers 1 and 4. These are the only constants available. Symbolic locations to be used are LOOPER, for sorting; LAST, for the test for completion; OUT, for counting flunks; and MAYBE, for counting marginal candidates.

Both

Problem 3.8. A warehouse records the number of months each vacuum tube has been in use, storing the information relative to TUBE. If a tube has been used more than 6 months, it should be checked; if it has been used more than 12 months, it should be replaced. Write a program to count the number of tubes in each category, placing the answers in CHECK and REMOVE. The total number of tubes is in TOTAL, which can be saved in COUNT. Locations ONE and SIX contain a 1 and a 6. Symbolic locations are SORT, for sorting; TEST, for the test for completion; and LOOK, to count tubes to be checked.

CLA HATS+1
STR GLOVES+1

PART FOUR: MULTIPLE ADDRESS MODIFICATION



Multiple Relocation

What two instructions relocate HATS into GLOVES?

Assume you need to relocate not just HATS but the entire series relative to HATS, putting it in the series relative to GLOVES. In other words:

HATS → GLOVES
HATS+1 → GLOVES+1
HATS+2 → GLOVES+2
HATS+3 → GLOVES+3
and so on.

The data processing portion of a looping program might be:

REPEAT CLA HATS
 STR GLOVES

But what instructions do you want performed on the second loop?

What addresses must be modified before the second loop begins—HATS, GLOVES, or both?

In brief, you want:

REPEAT CLA HATS
 STR GLOVES
 test for
 completion

modify HATS
modify GLOVES

 TRU REPEAT

You can modify HATS by working with its symbolic location ____.

Can you modify GLOVES the way the program is blocked out?

If not, give it a symbolic location so it can be modified. We will use MODIFY.

If the series relative to HATS has as many locations as the number in COUNT, location COUNT can be used as the loop counter and the program (without program preparation) would look as follows:

REPEAT	CLA HATS	Relocate HATS, copying it into
MODIFY	STR GLOVES	GLOVES
	CLA COUNT	
	SUB ONE	Check off the relocation just
	STR COUNT	completed
	TRZ STOP	
	CLA REPEAT	
	ADD ONE	Change HATS to HATS+1
	STR REPEAT	
	CLA MODIFY	
	ADD ONE	Modify GLOVES to GLOVES+1
	STR MODIFY	
	TRU REPEAT	Go back and relocate HATS+1
		into GLOVES+1
STOP	HLT	

A comment on cleaning out garbage from answer locations:

When a number is added to an answer location, that answer location must start with a zero. In adding COST into VALUE, for example:

```
CLA COST
ADD VALUE
STR VALUE
```

VALUE must start with a zero.

However, when a location is first used with the STR command, it need not be zeroed during program preparation. This is because the STR command automatically clears, or zeroes, the location before copying in the number from the accumulator.

You may remember an earlier problem where one number was repeatedly added into BIG, another number was repeatedly added into LITTLE, and then BIG and LITTLE were added together and stored in TOTAL with the instructions:

REPEAT

no

```
CLA BIG
ADD LITTLE
STR TOTAL
```

Answer locations BIG and LITTLE had to be zeroed during program preparation because they were first used with the ADD command, but not TOTAL since STR automatically zeroes it before copying in BIG+LITTLE.

Relocation problems present a similar situation, since all answer locations first appear in the program with the STR command.

Problem 4.1. Relocate the contents of the series COST through COST+99 into the series COST+100 through COST+199. Locations K1 and K100 contain a 1 and a 100, respectively. Use FIRST and SECOND as symbolic locations.

Problem 4.2. Relocate the series COST through COST+99 into even-numbered locations of the series relative to VALUE, zeroing out odd-numbered locations. CON1, CON2, and CON100 contain 1, 2, and 100. Use FIRST, SECOND, and THIRD as symbolic locations. What you want is:

```
COST    → VALUE
          VALUE+1 ← ZRO
COST+1  → VALUE+2
          VALUE+3 ← ZRO
          and so on.
```

So you would put COST into VALUE and zero into VALUE+1, changing COST to COST+1, VALUE to VALUE+2, and VALUE+1 to VALUE+3.

Multiple-Word Processing

A record, you may recall, is a series of numbers stored in consecutive locations of an address arithmetic series. For example, you have become accustomed to such phrases as "A warehouse maintains a five-word record on each vacuum tube in stock, stored relative to TUBE."

So far, you have worked only with one word of a record, such as TUBE+2. Now, however, you will work with more than one word, say TUBE+2 and TUBE+3. This means that both will have to be changed during address modification, making it another type of multiple address modification.

For example, a university maintains a three-word record on each student, stored relative to PUPIL: sex, class, and admissions score. Sex is coded by a 1 for male and a 0 for female. The university wants the total of all admissions scores for male students only.

How many words of the three-word record are needed to get the answer?

You will need the first word (PUPIL) to eliminate the females, and also the third word (PUPIL+2) to add up the admissions scores of the males, or the students who are left after sorting.

In brief, the problem requires you to sort, using the first word, and add, using the third word.

If TEST is the symbolic location for the test for completion, what are the sorting instructions?

Having eliminated the females, you then want to add the admissions scores of students that are left, the males. If TOTAL is the location for the sum, what are the adding instructions?

You have used two addresses, one to sort and one for adding, so both must be modified. What are the two addresses to be modified?

Here's the complete program, minus program preparation. ALL contains the total enrollment. THREE contains a 3.

LOOPER	CLA PUPIL	Sorting eliminates girls (the 0's)
	TRZ TEST	by skipping directly to the test for completion
MODIFY	CLA PUPIL+2	
	ADD TOTAL	Total up the admissions scores of
	STR TOTAL	students left, the males
TEST	CLA ALL	
	SUB ONE	The test for completion, which
	STR ALL	counts down total enrollment
	TRZ STOP	
	CLA LOOPER	
	ADD THREE	Get the first word of the second
	STR LOOPER	student
	CLA MODIFY	
	ADD THREE	Get the third word of the second
	STR MODIFY	student
	TRU LOOPER	Go back and process the second
		student
STOP	HLT	<i>two</i>

Assume that the first student in the problem above was a girl in the sophomore class with a score of 78, and the second student was a senior male with an admissions score of 95. Here are their records and what the program does with them:

	<u>Information</u>	<u>Data Processing</u>	<u>TOTAL</u>	
CLA PUPIL	PUPIL 0	A male? No. Check the		
TRZ TEST	PUPIL+1 2	next student.		
	PUPIL+2 78		0	
	PUPIL+3 1	A male? Yes. <u>Add in the</u>		
	PUPIL+4 4	admissions score and		
	PUPIL+5 95	check the next student	95	CLA PUPIL+2
				ADD TOTAL
				STR TOTAL

PUPIL &
PUPIL+2

Generally, the best place to start working on a problem is the data processing portion. Above, we sorted using the first word and added the third word of records not eliminated. With this figured out, the rest of the program—program preparation, the test for completion, and address modification—fell into place fairly easily.

We can't acquaint you with all the different types of problems requiring multiple-word processing, but we can give you several examples.

For example, a battle group codes men present for duty with a 1 and men absent with a 0, storing the information relative to MAN. Write a program to (1) count the men present for duty, placing the answer in READY; and (2) relocate these addresses in the series relative to DUTY.

Here's the sort of relocation you want:

CLA PLAYER
SUB OUT3
TRZ SHOWER

MAN	0	DUTY
MAN+1	1	DUTY+1
MAN+2	1	DUTY+2
MAN+3	0	DUTY+3
MAN+4	1	DUTY+4

If MAN has a 0, go right to the test for completion and then get MAN+1 and DUTY+1 ready for the next loop. However, if MAN has a 1, copy it into DUTY before running the rest of the loop.

If the test for completion starts at symbolic location FINAL, what are the sorting instructions?

If the program does not transfer, there must be a 1 in the accumulator. What instruction will copy this 1 into DUTY?

CLA PLAYER
ADD ERROR
STR ERROR

PLAYER &
PLAYER+1

What addresses must be modified before looping?

Problem 4.3. A library maintains a daily record of whether or not each of its books is checked out. The information is coded by a 1 if it is checked out or a 0 if it is not, stored relative to BOOK. The total number of books in the library is in VOLUMS. CHECK is to be used as the loop counter. Write a program that will count the number of books checked out, placing the total in OUT, and relocate the books checked out relative to FINE. Assume at least one book in the library. Location K1 contains a 1. Use READER for looping, MARK for address modification, and DONE for the test for completion.

Here's a slightly different type of problem. A baseball statistician keeps a three-word record on each professional baseball player-league, errors, and batting average-stored relative to PLAYER. League is coded by a 1 for American League, 2 for National League, and 3 for minor league. He wants a program to compute total errors made by all major league (American and National) players, storing the total in ERROR. OUT1 and OUT3 hold a 1 and a 3; SHOWER starts the test for completion.

You want to eliminate minor league players. Since minor league is coded with a 3, sorting should result in an immediate transfer to the test for completion if the first word (PLAYER) has a 3. What are the sorting instructions?

Here's how they work:

CLA PLAYER	Copy in a 1, 2, or 3.
SUB OUT3	Make it a -2, -1, or 0.
TRZ SHOWER	Skip the rest of data processing if it's a 0, since that indicates a minor league player.

Now you are ready to add up the errors, since only major league players are left. What are the adding instructions?

CLA MAN
TRZ FINAL

STR DUTY

Notice from the answer that adding uses a different word from the one used for sorting. What addresses must be modified before starting another pass?

MAN & DUTY

Get the data processing portions of the program straightened out and the remaining blocks of instructions will usually fall into place.

Problem 4.4. A supply depot maintains a two-word record for each vacuum tube in stock-cost and location-stored relative to TUBE. The depot has decided the information on location of each tube is not necessary and wants to eliminate it, converting to a one-word record instead. TOTAL contains the number of tubes in stock. COUNT is available for storage. Locations ONE and TWO contain a 1 and a 2, respectively. FIRST and SECOND can be used for address modification. Assume at least one tube.

The result should be a series of locations half as long as the original, with each location containing the cost of a different tube.

Problem 4.5. The Safety Officer maintains a three-word record for each officer on post. The words are: rank, number of tickets, and license number, stored relative to DRIVER. Rank is coded by a 1 for company grade, 2 for field grade, and 3 for general grade. Write a program to compute the total number of tickets issued to officers of company grade, storing the result in FINE; and the total number of tickets issued to field grade officers, storing the result in WARN. Assume at least one officer on post, the number in TOTAL. Location HOLD can be used for storage. Locations K1, K2, and K3 hold the numbers 1, 2, and 3, respectively. Symbolic locations are: TICKET, for looping; OVER, for the test for completion; BAR, for company grade officers; and LEAF, for field grade officers.

PART FIVE: REVIEW AND PRACTICE

Address Modification

Addresses can be changed by working with their symbolic locations. The format is:

clear-and-add the symbolic location
add the number necessary to get to the
next desired address
store back in the same symbolic location

If the cost of a tube is the second of five words in a five-word record relative to TUBE and the total cost of all tubes was needed, you would start with TUBE+1 (the second word) and add 5 to its symbolic location during address modification, to get to TUBE+6 for the cost of the next tube.

Problem 5.1. Write a program to clean out garbage from all memory locations from TRASH through TRASH+80. Use SAVE for temporary storage of the total number of locations to be zeroed. KON contains a constant of 1. ITEM contains the number 81. Use NEXT for address modification.

Problem 5.2. The procedure for computing the primary MOS involves getting a score (X) and then applying the formula $MOS = X + 1$. Compute $X + 1$ for each of 11 men. The X scores for each man are stored in locations relative to MAN. Store answers in TEMP to TEMP+10. NUM has a 12 and ONE has a 1. HOLD can be used for temporary storage. Use FIRST for looping and TOTAL for address modification. Be careful how you load the loop counter.

Sorting Techniques

Sorting is performed with TkZ, TRN, or both. The trick to sorting is making the number to be identified negative or zero so TRN or TRZ can pick it up, transferring outside the loop for further processing, if necessary. Locations not causing a transfer can be processed immediately after sorting instructions.

Problem 5.3. Find how many locations from SORT to SORT+75 contain positive numbers, storing the count in ANSWER. KON contains a 1 and TOTAL contains a 76. Use LAST for the completion test and ITEM for address modification.

Problem 5.4. An electronics depot maintains a six-word record on each vacuum tube in stock, starting at location SUPPLY. The words are: tube function, part number, stock level, original cost, present cost, and storage location. ITEMS contains the number of tubes in the depot. TEMP is a temporary storage location for the number of tubes. The stock level word is coded with a 1 if the tube should be reordered, a 2 if the stock level is satisfactory, and a 3 if the amount on hand exceeds requirements. The depot wants a program that will determine the number of tubes in the excess category and the number in the shortage category, storing the counts in NEED and EXCESS. Do not assume at least one item is in stock. Location KON1 has a 1, KON2 has a 2, and KON6 has a 6. Use COMPUT to allow modification of stock level word, DONE for the test for completion, and ORDER for counting.

Problem 5.5. The drinking age in a town is 21. The Alcoholic Beverages Commission wants to know how many people in the town are of drinking age now and how many will be of drinking age next year so they can estimate from potential income tax how much raise to give the Commissioners. The town has a population of 1,000, which is the number in TOWN. The ages of the 1,000 people are stored in sequential locations starting at location DATA. Store the number of people who are of drinking age now in location NOW, and the number who will be of drinking age next year in LATER. CON1 contains a 1 and CONST has a 21. Use COMPUT for modifying addresses containing ages, TEST for the completion test, and NEXTYR to check for persons who will be 21 next year.

Multiple-Word Processing

Sometimes more than one word of a record is needed. Items may be sorted on the basis of one word (TUBE+2, for example) and the cost of those falling into one pile computed using a different word (such as TUBE+4). No matter what the problem, it is always best to start working on the data processing portion first.

Problem 5.6. Some enlisted men have taken screening tests in application for OCS. The information is stored in memory relative to APPLY. There is a five-word record for each man giving the following information: serial number, rank, date of application, test score, and acceptance or rejection (a 1 for acceptance and a 2 for rejection). Write a program to determine the number who were accepted, and the total points for those who were accepted, storing the answers in PLUS and VALUE, respectively. Location OFFCAN contains the total number of applicants. Use MEN as temporary storage for this number. KON contains a 1, KON2 a 2, and KON5 a 5. Use DATA for modifying addresses containing test status, ACCEPT for modifying addresses containing test scores of those accepted, and LAST for the completion test. Use the TRN command for sorting.

- 101 -

PHASE IV

ADVANCED TECHNIQUES

This section deals with advanced transfer techniques in computer programming. These are: new methods of address arithmetic; the use of index registers; and the use of three new commands--LOD, TRX, LDX.

You will learn to transfer by use of a symbolic location word plus address arithmetic. For instance: TRU TOTAL +2, which would say to the computer to go to the second step after the symbolic location word TOTAL. The other method you will learn is the use of the asterisk (*) for transferring. This method also uses address arithmetic. For example: TRU *-1, which says to transfer from this step (TRU *-1) back one step, or go to the step just before the present one.

In this section you will learn to use index registers as places for temporary storage of numbers. The index registers are used in the test for completion and in address modification to save the original address for use later.

The new commands--LOD, TRX, LDX--are simply methods to shorten a program. They can, as you will find later, do several instructions at the same time. The LOD command will load numbers into index registers. The TRX command will modify two index registers at the same time, increasing the one used for address modification and decreasing the one used in the completion test. The LDX command allows you to load two index registers at the same time, but it has one limitation. You must know the exact number of "loops" or "passes" needed for a computation before using LDX.

PART ONE: TRANSFER TECHNIQUES

Preview. Transfers can be made to take place by writing an asterisk (*) with a number indicating the next instruction to be performed, relative to the present one. TRU *+2 means transfer to the instruction two places ahead. TRZ *-1 means repeat the last instruction if the accumulator is zero. (TRZ--transfer if zero).

Instructions needed in a transfer can also be indicated by their position relative to a symbolic location. TRU REPEAT-1 means go to the instruction occurring just before symbolic location REPEAT.

The Asterisk with Address Arithmetic

Let's say you have a program whose last three instructions are:

```

                TRZ STOP
                TRU REPEAT
        STOP    HLT

```

The HLT command is how many instructions ahead of the TRZ STOP instruction?

Symbolic location STOP can be eliminated completely by substituting an asterisk with a +2, to indicate the instruction two places ahead. The program would then look like this:

```

                TRZ *
                TRU REPEAT
                HLT

```

The asterisk is used to refer to an entire instruction and not just an address.

The asterisk with address arithmetic is also useful in address modification. Modify COST in the sequence below to COST+1, assuming there is a 1 in location ONE.

```

        CLA *
        ADD ONE
        STR *
        CLA COST

```

Notice that the numbers change as the program steps forward, from +3 to +1.

Rewrite the following program in your head, using the asterisk. Instructions that should be changed are numbered. Answers are on the opposite margin.

	REPEAT	CLA COST
		ADD VALUE
		STR VALUE
		CLA COUNT
		SUB ONE
		STR COUNT
1.		TRZ STOP
2.		CLA REPEAT
		ADD ONE
3.		STR REPEAT
4.		TRU REPEAT
	STOP	HLT

CLA THIS

Problem 1.1. There are two kinds of vacuum tubes in an inventory, 6SN7 tubes and 6AQ6 tubes. Write a program to compute the total value of both kinds, placing the answer in TOTAL. The number of 6SN7 tubes is in STOCK1. The number of 6AQ6 tubes is in STOCK2. Temporary storage will be TEMP and TEMP+1. The cost of a 6SN7 is in VALUE and the cost of a 6AQ6 is in VALUE+1. Store the total worth of 6SN7 tubes in VALSTK and the total worth of 6AQ6 tubes in VALSTK+1. Assume at least one tube of each type. There is a 1 in KON. If you have to, use COMPUT as symbolic location for the first looping program, NEXT for the second, and SUM to get the total of both. Use asterisks wherever possible.

2

Problem 1.2. This problem is like the preceding one, except that the number of each type of tube is not known. Instead, each tube has information on its type stored relative to TUBE. Each location in the series has either a 1 or a 0--a 1 to indicate a 6SN7 tube, and a 0 to mean a 6AQ6 tube. The total number of all tubes is in STOCK, which can be saved in TEMP. Assume at least one tube. Use NEXT for sorting, LAST for the test for completion, and COMPUT for computation outside the loop. Use asterisks wherever possible. VALSTK and VALSTK+1 will not be needed. You can add VALUE into TOTAL in one subroutine and VALUE+1 into TOTAL in another.

+3

Symbolic Locations with Address Arithmetic

Tracing out a program that uses asterisks becomes a little difficult, especially for someone else, when the skips are much greater than 9 instructions. In these situations, you are better off using a symbolic location with address arithmetic.

TRU DONE+2 means, "Transfer to the instruction two steps after symbolic location DONE."

What instruction would be performed after TRU OK-1 below?

CLA THIS
OK STR THERE
TRU OK-1

1. *+5
2. *-7
3. *-9
4. *-10

	CLA MODIFY		CLA MODIFY+2
	ADD ONE	MODIFY	ADD ONE
	STR MODIFY		STR MODIFY+2
MODIFY	CLA COST		CLA COST

(cont.)

STOP

CLA INFO+5

LAST

CLA	COUNT
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	1
21	1
22	1
23	1
24	1
25	1
26	1
27	1
28	1
29	1
30	1
31	1
32	1
33	1
34	1
35	1
36	1
37	1
38	1
39	1
40	1
41	1
42	1
43	1
44	1
45	1
46	1
47	1
48	1
49	1
50	1
51	1
52	1
53	1
54	1
55	1
56	1
57	1
58	1
59	1
60	1
61	1
62	1
63	1
64	1
65	1
66	1
67	1
68	1
69	1
70	1
71	1
72	1
73	1
74	1
75	1
76	1
77	1
78	1
79	1
80	1
81	1
82	1
83	1
84	1
85	1
86	1
87	1
88	1
89	1
90	1
91	1
92	1
93	1
94	1
95	1
96	1
97	1
98	1
99	1
100	1



Use as few symbolic locations as possible in the following problems by (1) using the * for transfer of 9 steps or less, and (2) address arithmetic relative to a symbolic location for transfers of more steps.

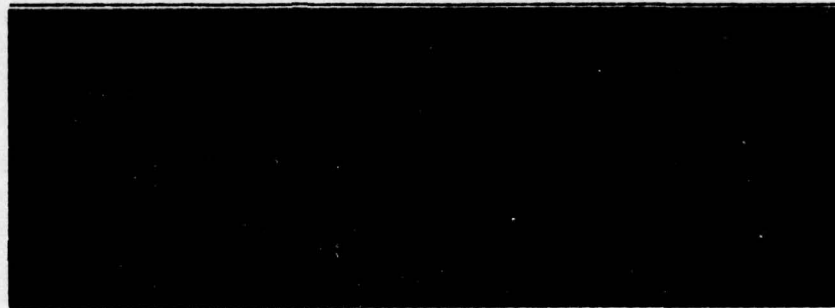
Problem 1.4. There is information about the ages and ranks of enlisted men and officers starting at location INFO, in the following three-word format: enlisted man or officer, age, and rank. The first word is coded with a + to indicate an officer, or a - to indicate an enlisted personnel. Count the number of enlisted personnel in location TOTEM. The total number of personnel is in location MEN. A constant of 1 is in K1 and a constant of 3 is in K3. Use EM as symbolic location for counting. Use DATA to modify the record to be processed. (The + and - signs can be considered the same as positive and negative numbers.)

Problem 1.5. Headquarters wants to know the number of second lieutenants eligible for promotion to First Lieutenant at the end of this month. Eighteen months active duty in grade are required for promotion from Second Lieutenant to First Lieutenant. The data for officers are stored in 5-word records as follows: rank, serial number, months in grade on active duty, MOS, and assigned unit. Ranks for officers are coded by 1 for Second Lieutenant, 2 for First Lieutenant, and so on. The records are kept relative to RATING. PERSON contains the number of personnel. This number is not to be destroyed. Store it temporarily in HOLD. Assume at least one officer. An 18 is in TIME, a 1 is in KON, and a 5 is in K5. Store the number of eligible officers in UP. Use RANK to modify addresses referring to rank, LAST for the completion test, and SECOND for instructions referring to time in grade if the officer is a Second Lieutenant.

Problem 1.6. As a programmer in the logistical accounting office of Base Headquarters, you are given an assignment to write a program to provide the Management Branch with a Statistical Summary Report covering their weekly activities, which consist of keeping Memos for the Record, Requisitions, Advances, etc. The program is to obtain separate totals for the amount of each kind of activity per week. These amounts will vary from week to week. The information is stored relative to memory location STAT. When the end of an activity has been reached, there will be simply a negative, or minus sign, in the memory location addressed. This will mean that the previous accumulated amount represents a total for an activity. Store all totals relative to TOT. The number of totals to be produced for this week's report is in COUNT. There may not be any totals this week. A constant of 1 is in K1. Use NEXTOT and AGAIN to modify addresses.

Problem 1.7. An insurance company's policy records of 50 men are stored in consecutive 3-word groups, starting at location RECORD. Each man's record contains the following items: policy number, total annual premium, and premiums paid to date this year. Form a fourth record word for each man and relocate the 50 4-word records relative to UPDATE. The fourth word is to contain the amount of premium remaining to be paid. The number 50 is in COMPY, which should be stored temporarily in HOLD. Use all three constants: K1 = 1, K3 = 3, and K4 = 4. OLD is the symbolic location for modifying addresses in expanding and relocating the record. After completing the above, write a second program to compute the total amount to be collected for the remainder of the year and store the amount in TOTAL. Remember that the 4-word records are now stored relative to UPDATE. Use NUM as temporary storage for COMPY, and SUM for address modification in obtaining the total remaining to be collected. Only two constants are required: K1 and K4.

PART TWO: INDEX REGISTERS



Introduction to Indexing

The special temporary locations used for this new type of address modification are called index registers. The procedure for using them is called indexing. Only one new type of instruction is involved in indexing. The basic form of that instruction is shown below:

ADD DATA,IR2

The new parts are the comma after DATA (,) and the address for the index register number (IR2). All that remains is to explain what the instruction does and where to use it.

Say that you are adding all information stored relative to DATA (DATA, DATA+1, etc.). You have added the contents of DATA on the first pass and want to add the contents of DATA+1 on the next pass. If there is the number 1 located in IR2, the following instruction will do it:

ADD DATA,____

CLA, STR, ADD,
SUB --- TRZ,
TRN, TRU

In effect, ADD DATA,IR2 says to copy DATA into a special temporary location (IR2) and there add to it the contents of that location (IR2). As IR2 contains the number 1, DATA+1 results. Thus, the complete instruction says the same thing as: ADD DATA+1.

Problem 2.1. Write an instruction that says the same thing as SUB COST+1. Index Register 3 contains the number 1.

HLT

Problem 2.2. Write an instruction that says the same thing as STR BOOK+2. Index Register 1 contains the number 2.

The Comma

Why the comma?

It is needed to separate two address fields, the one that we have been using all along and a new one that is used in indexing.

Up to now, you have been working with symbolic location fields, operational codes (the commands), and a single address field. For example:

	<u>Symbolic Location</u>	<u>Op. Code</u>	<u>Address Field</u>
STR DATA+4	REPEAT	ADD	COST

The address field you have been using is called the First Address Field. All addresses that we have used previously appeared in the First Address Field.

Indexing uses another address field, known as the Second Address Field. And, thus, the comma. Anytime more than one address field is used, a comma is required to keep them separated. Hence the comma:

ADD DATA,IR2

What is the address field for DATA in the instruction above? IR2?

Any command (operational code) used up to now, except HLT, can be used with indexing. These commands, by way of review, are: _____, _____, _____, _____, and the transfer commands _____, _____, _____. Since these commands--and only these commands--are used in indexing, they are called indexable commands.

Which of the following are not indexable commands?

TRU	STR	HLT	TRZ
ADD	SUB	TRN	CLA

For completeness, you should know that index registers do not have a plus or minus sign, as the accumulator and standard memory locations do. Also, if you haven't already guessed, they can modify only when written in the Second Field. Then they change what appears beside them in the First Address Field. In other words, you would never write anything like IR3,COST. An instruction like that would always be written the other way, as COST,IR3 -- with the index register always appearing in the Second Address Field.

Now let's see what type of instructions are used and what they do.

STR DATA,IR2

The instruction copies DATA into a temporary location (IR2) and there adds the contents of Index Register 2. Then whatever is in the accumulator is stored (STR) at the modified address. If IR2 contains the number 4, for example, the instruction indicates _____.

To say the same thing in more precise language, the first address (DATA) is increased by the amount in the second address (IR2). The number in the accumulator is then stored (STR) in the modified first address. All of this modification takes place at a temporary location so that the original address (DATA) is not destroyed.

TRU TUBE,IR4

In the instruction, TRU is an indexable command. TUBE is written in the First Address Field and IR4 in the Second Address Field. The entire instruction says to branch unconditionally (TRU) to the location indicated by the sum of the First and Second Address Fields, but not to modify or change the first address TUBE in any way.

First,
Second

Now let's take the block of instructions which modify an address and see how we use the index register to simplify the job of modification. As you will remember, we modified an address by using the block of four instructions below (using REPEAT as the symbolic location and COST as the address):

CLA REPEAT

```

____
____
____

```

This procedure added a one to the address, giving COST+1 on the next pass through the program. However, by the use of index register we save the address COST by using the temporary storage: the index register.

The first step is to zero the index register that we are going to use. This is done easily enough by:

CLA ZRO
STR IRI

Now suppose we want to add the contents of COST, COST+1, COST+2, etc., into TOTAL.

The IRI already has a zero. So to ADD COST and to modify our address to pick up a new number each pass we use the following instructions:

REPEAT ADD COST,IRI
STR TOTAL

Since IRI contains a zero, the instruction says to CLA COST+0, or simply CLA COST. Now we want to modify COST so that on the next pass the computer will pick up COST+1. This is done by the block of instructions that follow:

CLA IRI
ADD ONE
STR IRI
TRU REPEAT

ADD COST,IRI

Since the index register (IRI) had a zero, our adding one to it gives it a total of one, so that when we transfer (TRU) back through (REPEAT ADD COST,IRI) we have added COST to the contents of IRI (a "1"), giving us COST+1; the next time COST+2, and so on each pass. This will bring up the contents of COST, COST+1, etc., each time, leaving the original location intact.

CLA IRI
ADD ONE
STR IRI

ADD ONE
STR REPEAT
TRU REPEAT

When an index register is stuck on after a first address (CLA COST,IR1), as IR1 here, it is often called a tag. With index registers "tagged" on behind, as they are, the whole process of indexing is often referred to as tagging.

Now we are ready to use indexing in a complete program. Find the total of the numbers in COST through COST+9, placing it in TOTAL. COUNT contains a 10.

Old Program

New Program

```

CLA ZRO
STR IR1
REPEAT
STR TOTAL
CLA COUNT
SUB ONE
STR COUNT
TRZ STOP
STR TOTAL
TRU REPEAT
STOP
HLT

```

Now, check your understanding of the new program with the following explanation:

CLA ZRO	First we zero IRI so that
STR IRI	the first address of any instruction "tagged" by the index register will not be increased on the first pass through the loop.
REPEAT ADD COST,IRI	Since IRI contains a zero, the instruction says to CLA COST+0, or simply CLA COST. Said a little differently, the first address is tagged with a zero.
CLA COUNT	This is the old familiar test
SUB ONE	for completion.
STR COUNT	
TRZ STOP	
CLA IRI	Instead of modifying the first
ADD ONE	address in the processing instruction (CLA COST,IRI), we
STR IRI	add a 1 to the index register each time. This keeps the first address COST from being changed,
CLA TOTAL	Pick up the running sub-total
TRU REPEAT	and branch back to the top of the loop.
STOP HLT	

Check this over a few times. Practice problems appear on the next few pages.

Indexing and the Test for Completion

Index registers can also be used as temporary storage locations in the test for completion. The number to be used in the loop counter is copied into the index register as part of program preparation, just as you have done before.

However, rather than check for zero items in program preparation, we will add the check down in the test for completion instead. This is a little different from what you are used to, so examine the new program closely. It will be useful in more advanced program writing.

<u>Old Program</u>		<u>New Program</u>	
	CLA COUNT		CLA COUNT
	STR TEMP		STR IR2
	TRZ STOP		-----
	CLA ZRO		CLA ZRO
			STR IR1
AGAIN	ADD TUBE	AGAIN	ADD TUBE,IR1
	STR TOTVAL		STR TOTVAL
	CLA COUNT		CLA IR2
	--- ----		TRZ STOP
	SUB ONE		SUB ONE
	STR COUNT		STR IR2
	TRZ STOP		TRZ STOP
	CLA AGAIN		CLA IR1
	ADD ONE		ADD ONE
	STR AGAIN		STR IR1
	CLA TOTVAL		CLA TOTVAL
	TRU AGAIN		TRU AGAIN
STOP	HLT	STOP	HLT

Now turn to the next page for a desk check of the complete program.

DESK CHECK OF COMPLETE PROGRAM

Assume that COUNT contains the number 2.

CLA COUNT	Since COUNT contains a 2, storing it in IR2 gives IR2 a 2 also.
STR IR2	
CLA ZRO	Zero IR1.
STR IR1	
AGAIN ADD TUBE,IR1	Add the cost of the first tube (TUBE+0). Store it in TOTVAL, making TOTVAL equal to TUBE.
STR TOTVAL	
CLA IR2	Test IR2 for a zero. Since it has a 2, the program continues on to the next step.
TRZ STOP	
SUB ONE	Decrease the counter (IR2) by one for this pass. IR2 now equals 1. Test for zero again. Since IR2 contains a 1, the program continues on to the next instruction.
STR IR2	
TRZ STOP	
CLA IR1	Add a 1 to IR1, changing it from zero to 1.
ADD ONE	
STR IR1	
CLA TOTVAL	Pick up the running subtotal, TOTVAL, and go back for another loop.
TRU AGAIN	

AGAIN ADD TUBE,IR1	As we begin the loop for the second time, the accumulator has the contents of TOTVAL (which is equal to TUBE on the first pass). Since IR1 has been changed from zero to 1, TUBE+1 is now added in.
STR TOTVAL	TOTVAL now equals the contents of TUBE plus TUBE+1.
CLA IR2	Since IR2 now contains a 1, the TRZ command does not result in a transfer. Subtracting 1, however, now reduces IR2 to zero, so that the TRZ STOP instruction now halts the program.
TRZ STOP	
SUB ONE	
STR IR2	
TRZ STOP	

STOP	HLT

Problem 2.4. A soap company records the sales made by each salesman, stored relative to SALES. That is, the number of sales made by the first salesman is in SALES, the number made by the second is in SALES+1, and so on. Write a program to compute the total sales of all salesmen, placing the answer in TOTAL. The company employs 16 salesmen. There is a 16 in MEN, which can be saved by storing it temporarily in TEMP. There is a 1 in location ONE. Use SELL as symbolic location for looping. For address modification, index register #2 (IR2) is available.

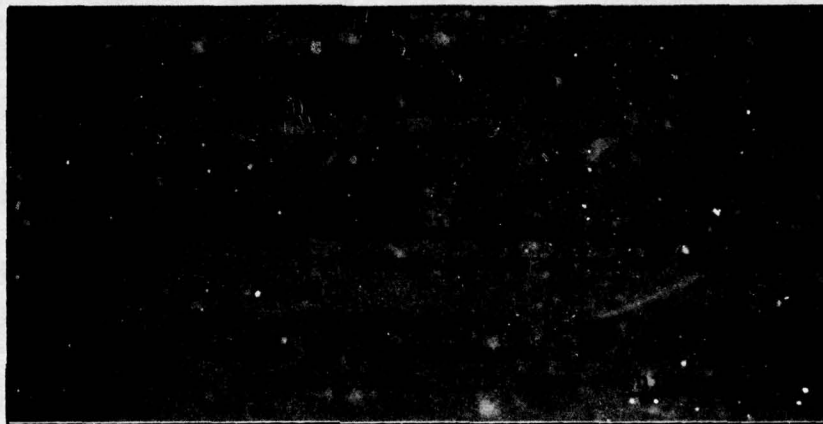
Problem 2.5. A large music store sells records of many different types, such as classical, western, folk, and so on. The number of different types is in TYPES. The sales of each type record is stored in sequence relative to RECORD. Write a program to compute the total number of records sold of all types, placing the answer in SALES. Use POP as storage for the number of types (for the loop counter) and HIT as the symbolic location for looping. There is a 1 in location K1. Use index register #1 (IR1) for address modification.

Problem 2.6. The pay record for each man in a business firm has information in the following order of words: badge number, shift worked, regular pay, and overtime pay. These words are stored relative to RECORD. The total number of employees is in HELP. Compute the total regular pay for all employees, placing the amount in REGPAY. There is a 1 in CON1 and a 4 in CON4. Use IR4 for temporary storage and TOTAL as symbolic location for looping. For address modification, use index register #3 (IR3).

PART THREE: THE LOD COMMAND



LOD VALUE,,IR4



Third

2

2

Now you will learn the use of the new command LOD, which, with one instruction, permits the loading of an index register. The instruction is of the following form:

LOD COST,,IR2

The new parts are the command itself (LOD) and the two commas (,,).

To understand what LOD command does and how it works, we will have to expand the address field. In addition to the first and second fields, there is a third field.

	<u>First</u>	<u>Second</u>	<u>Third</u>
LOD	COST	,,	IR2

The LOD command can be used to load an index register. When it is used this way, the location containing the number to be loaded is written in the first field, and the index register in the third field.

Since the Second Address Field has not been used at all, the omission is indicated by the double comma. (If the fields were side by side, only one comma would be needed.)

The LOD command copies the contents of the memory location (COST) into an index register (IR2).

Write an instruction to copy the contents of VALUE into IR4: _____. IR4 is in the _____ Address Field. Memory locations such as COST and VALUE are never written in the third field.

The LOD command works like a STR command. That is, the number that is copied, the number in the location indicated by the first address, is not changed or destroyed. But anything in the index register specified by the third address is destroyed and replaced by the number from the first address.

Say that HOUSE contains the number 2 and IR4 contains a zero. After the instruction LOD HOUSE,,IR4 -- HOUSE will contain a _____, because that number is merely copied, not destroyed. But IR4 will now contain a _____ rather than a zero.

Try this: AUTO contains a 4 and IR2 a 6. What will be in AUTO and IR2 after the instruction LOD AUTO,,IR2?
_____, _____.

The LOD command is an indexable command because it involves index registers in the modification of addresses. An example of how it is used is shown below.

Without the LOD Command

CLA COUNT
STR IR2

With the LOD Command

LOD COUNT,,IR2

Without the LOD command, two instructions are required to copy the contents of COUNT into IR2. With the LOD command, only one instruction is required.

The LOD command can also be used to zero index registers. What would the instruction be if you wanted to zero (ZRO) IR3 with the LOD command?

Note that we could not zero a memory location this way because memory locations cannot be written in the third field. For example, you could not write an instruction such as LOD ZRO,,TOTAL.

For present purposes, only index registers can be addressed in the third field. Try a few instructions for practice, using the LOD command.

Problem 3.1. Place a zero in IR1.

Problem 3.2. Copy the contents of MAN into IR2.

Problem 3.3. Replace the number in IR3 with the number in COUNT.

Now we will rewrite some program preparation instructions using the LOD command:

Old Way

CLA ZRO
STR IR1
CLA COUNT
STR IR2

New Way

LOD ZRO,,IR1
LOD COUNT,,IR2
CLA ZRO

With the LOD command we load index registers without disturbing the contents of the accumulator. So why is CLA ZRO the last instruction under the "new way"? If we use the LOD command to load IR1 and IR2, we still need to zero the accumulator before the program carries out the next instruction in sequence.

Using the new command, LOD, we need a new address field, the third. There are two commas in an instruction using LOD to show the omission of the second field.

LOD ZRO,,IR3

The LOD command, such as:

LOD COST,,IR2

says to load the contents of the first address, COST, into the third address, IR2. The use of LOD is a shorter, more efficient way of programming.

Problem 3.4. A publishing company maintains a 3-word record on each of its books, stored relative to BOOK. The words are: date of publication, retail price, and number of sales. The number of books is in TOTAL. Using IR1 for looping and IR2 for address modification, write only the program preparation for this problem, using the LOD command.

Problem 3.5. You have 10 items stored relative to TIE. The total number of items (10) is in TEN. Write a program to add TIE through TIE+9, placing the answer in ANSWER. Use IR1 for temporary storage in the test for completion, and IR3 for address modification. Use CRAVAT as symbolic location for looping. Location ONE contains a 1.

Problem 3.6. Each vacuum tube in stock has information in its record in the following order of words: brand name, date last used, place last used, and hours used. These words are stored relative to TUBE. The total number of tubes in stock is in ITEMS. Use IR1 as temporary storage for this total. Write a program to compute the total number of hours all tubes have been used and store this value in HOURS. There is a constant of 1 in ONE and a 4 in FOUR. Use IR2 for address modification and REPEAT as symbolic location for looping.

PART FOUR: THE TRX COMMAND

The TRX command is a conditional transfer command. It is used to transfer the program counter to a new address if a specified condition exists. The condition is specified by a code in the Op Code field of the command. The address to transfer to is specified by the three address fields of the command. The TRX command is used to transfer the program counter to a new address if a specified condition exists. The condition is specified by a code in the Op Code field of the command. The address to transfer to is specified by the three address fields of the command. The TRX command is used to transfer the program counter to a new address if a specified condition exists. The condition is specified by a code in the Op Code field of the command. The address to transfer to is specified by the three address fields of the command.

IR1

The TRX command is a conditional transfer command. It is used to transfer the program counter to a new address if a specified condition exists. The condition is specified by a code in the Op Code field of the command. The address to transfer to is specified by the three address fields of the command. The TRX command is used to transfer the program counter to a new address if a specified condition exists. The condition is specified by a code in the Op Code field of the command. The address to transfer to is specified by the three address fields of the command.

Introduction to TRX

The TRX command acts on the contents of an index register instead of on the contents of the accumulator. TRX stands for Transfer on indeX. That is, the program will transfer depending on what is in an index register rather than what is in the accumulator. Note that this command does not disturb the accumulator. TRX is a conditional transfer command. So you must remember that a conditional transfer command will first check to see if a specified condition exists before the program transfers.

The TRX command uses the Op Code and all three address fields. The command also links, or pairs, successive index registers.

Using the TRX Command

The basic form of a TRX instruction is as follows:

TRX *-1,IR2,1

First you will learn what such an instruction does. Then we will fit it into a program to show where and how it is used.

When the computer reaches a TRX instruction, such as the one shown above, the following things happen:

First, the next numbered index register following the one listed in the second field is called up. This is the paired index register. Since IR2 appears in the instruction, the next numbered index register would be IR3.

Remember, the TRX command pairs index registers; always the IR given in the instruction is paired with the next numbered IR: IR1 with IR2, IR2 with IR3, IR3 with IR4, and IR4 with _____. Also, for present purposes, we are only using four index registers; so the highest, IR4, when listed in the Second Address Field, is paired with the lowest, IR1.

Second, after the next numbered, or paired, index register is addressed (IR3 in the example), it is tested for completion; that is, it is tested for zero. Next, it is decreased by one and then retested for zero. Remember, in our example the index register tested would be IR3.

Finally, if on the second test for zero the paired IR is not zero, the number in the Third Address Field (in our example a 1) is added to the contents of the IR in the Second Address Field (IR2) and the problem transfers to the address given in the first address (*-1). So in our example, the one (in the third field) is added to the IR2 (in the second field) and the program transfers to the preceding instruction, *-1, shown in the first field.

If either test for zero finds a zero in the paired index register, the program does not transfer, but continues on to the next instruction.

The rest of this section merely gives an example for the material presented in this part.

To desk check a complete program, let us assume location COUNT contains the number 3 and locations VALUE, VALUE+1, and VALUE+2 contain \$200, \$300, and \$400, respectively. So our program is:

```

LOD COUNT,,IR2      (IR2, the loop counter)
LOD ZRO,,IR1        (IR1, the address modifier)
CLA ZRO
ADD VALUE,IR1
TRX *-1,IR1,1
STR VALSTK
HLT

```

Desk check:

<pre> LOD COUNT,,IR2 LOD ZRO,,IR1 CLA ZRO ADD VALUE,IR1 </pre>	<p>The LOD COUNT,,IR2 instruction will copy the number 3 from COUNT into IR2. LOD ZRO,,IR1 will zero IR1. CLA ZRO will zero the accumulator. When the ADD VALUE,IR1 instruction is carried out, the address VALUE is augmented by the contents of IR1. Because IR1 is zero, the contents of location VALUE is added to the accumulator, giving a sum of \$200--the accumulator was zero before VALUE was added to it.</p>
<pre> TRX *-1,IR1,1 </pre>	<p>The program now steps to the next instruction, TRX *-1,IR1,1. When this instruction is carried out, IR1 automatically is paired with IR2; then IR2 is tested for zero. IR2 contains the number 3. Since it is not zero, it is decreased by one (for this pass through the program), leaving a remainder of 2. IR2 is again tested for zero and because it is not zero the number in the third address (a 1) is added to IR1. IR1 now contains a 1.</p>

The TRX command now causes the program to transfer to the location specified in the first address, *-1. Remember that the asterisk (*) can be read as "this location..." So the program transfers to the preceding instruction, ADD VALUE,IR1.

Notice that the accumulator was not affected while the TRX instruction was being carried out. Thus, it still contains \$200 (the contents of VALUE) from our first computational pass.

Next, the program will again carry out the ADD VALUE,IR1 instruction. This time when the first address is augmented by the contents of IR1, which is a 1, the actual address of the instruction is VALUE+1. Thus, the contents of VALUE+1 (\$300) is added to the accumulator, which already contains \$200, giving a sum of \$500--the sum of the contents of locations VALUE and VALUE+1.

Now the TRX instruction is carried out a second time. (Note that the TRX command will not disturb the accumulator.) The first test for zero finds a 2 in IR2, so IR2 is decreased by one, leaving a 1 in IR2. Again, IR2 is tested for zero; because it is not zero, the number in the third address is added to IR1. Now IR1 contains a 2. The program again transfers to *-1, the address specified in the first field. For the third time the instruction ADD VALUE,IR1 is augmented by the contents of IR1--this time a 2. Thus, the actual address of the ADD instruction is VALUE+2. The \$400 from location VALUE+2 is added to the contents of the accumulator, which is already \$500. Thus, the accumulator now holds \$900--the sum of VALUE, VALUE+1, and VALUE+2.

Again, the TRX instruction is carried out. This time on the first test for zero, IR2 contains a 1. Because it is not zero, it is decreased by one, then retested for zero. On the second test for zero, IR1 contains a zero, so the program continues in sequence to the STR VALSTK instruction. This instruction puts the contents of the accumulator (\$900) into location VALSTK. The program steps to the HLT command and the computer stops.

Our program works when we assume location COUNT contains a 3; therefore, we know it will work for any number of items in stock.

Let's compare our stock program written two ways:

Program Using Only LOD
Command

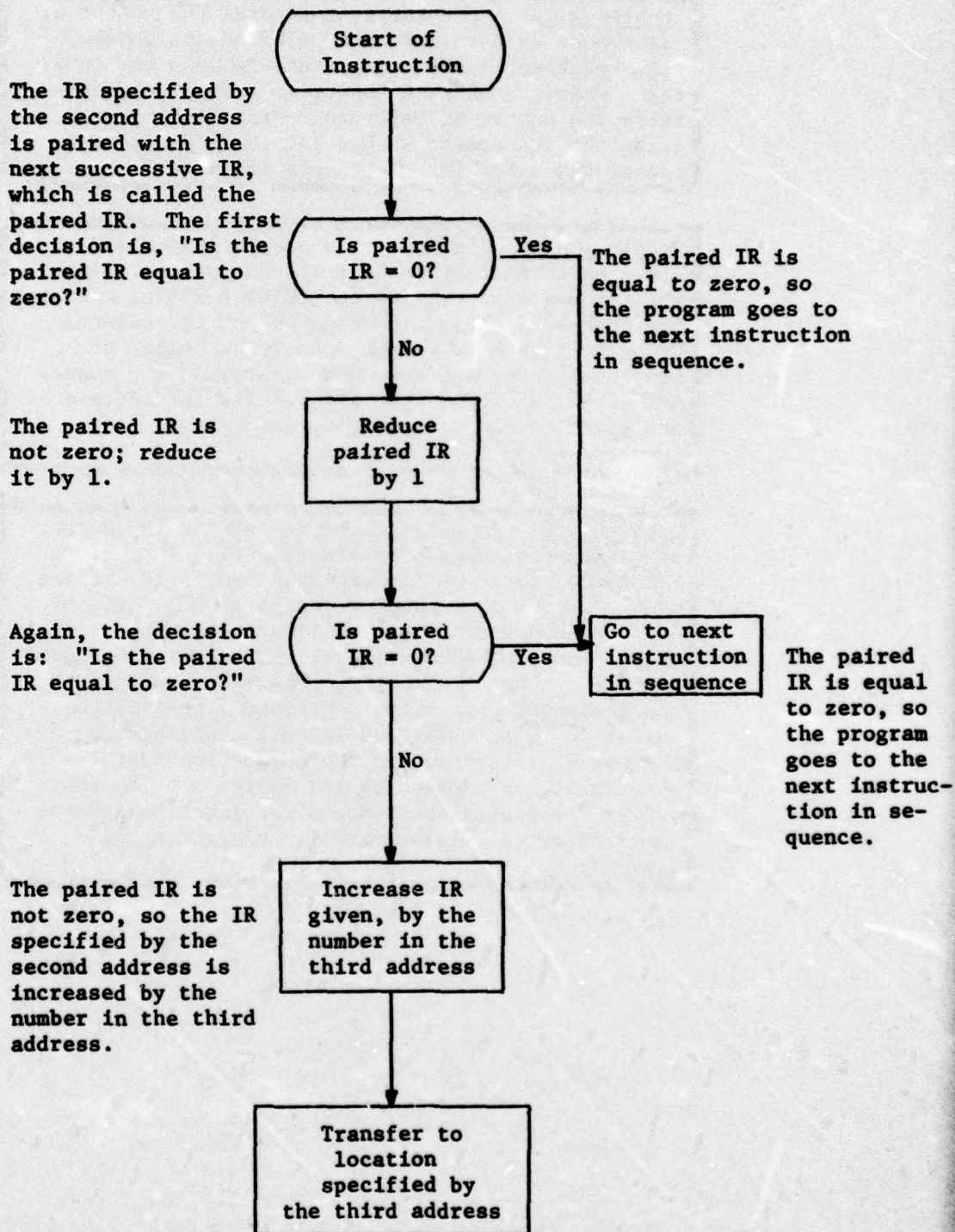
```
      LOD COUNT,,IR2
      LOD ZRO,,IR1
      CLA ZRO
COMPUT ADD VALUE,IR1
      STR VALSTK
      CLA IR2
      TRZ STOP
      SUB K1
      STR IR2
      TRZ STOP
      CLA IR1
      ADD K1
      STR IR1
      CLA VALSTK
      TRU COMPUT
STOP   HLT
```

Program Using LOD
and TRX Commands

```
      LOD COUNT,,IR2
      LOD ZRO,,IR1
      CLA ZRO
      ADD VALUE,IR1
      TRX *-1,IR1,1
      STR VALSTK
      HLT
```

You can see that by using the TRX command you can save writing _____ instructions in this simple program. This saving alone makes the TRX command extremely valuable in program writing. Although the command is somewhat complex, it greatly reduces the labor involved in constructing program loops and, because it takes fewer instructions, it will reduce the chances for errors in a program.

Because the TRX command is complex, let's review it again--this time using a flow chart to show its sequence of operation:



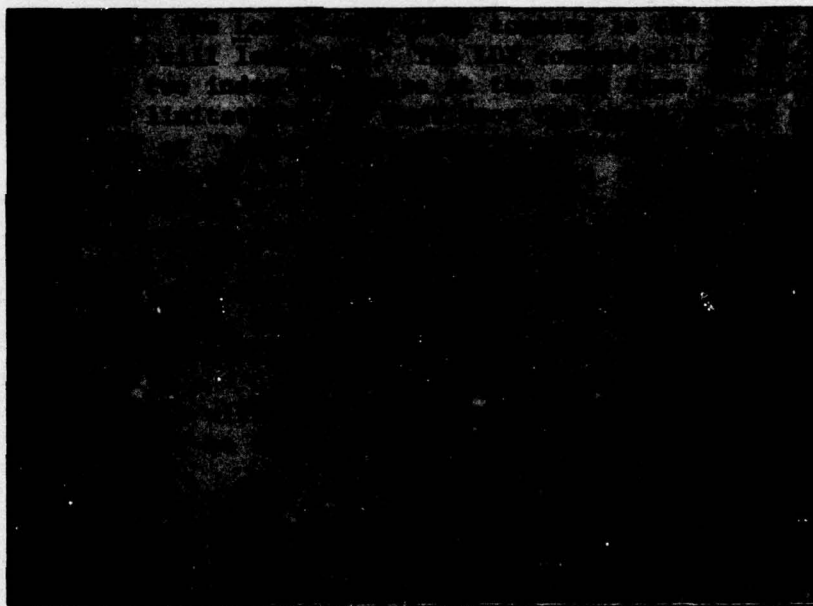
Problems

Problem 4.1. A manufacturer records the number of sales made by each salesman, storing the information relative to PUSH. He wants to know the total sales made. Store the answer in BONUS. MEN contains the number of salesmen. Write the program using the TRX command. Use IR1 for address modification and IR2 for the completion test.

Problem 4.2. A clinic records the number of house calls made by each of its doctors, storing the information relative to DOCTOR. MEDICS contains the number of doctors working out of the clinic. Write a program to compute the total number of house calls for the past month, storing the answer in CALLS. Use TRX. Use IR2 and IR3 for address modification and for the completion test, respectively.

Problem 4.3. An airline company stores the number of passengers on each of its flights. They store the number of in-passengers relative to IN and the number of out-passengers relative to OUT. Assuming the number of flights arriving each day is in ARRIVE and the number leaving each day is in LEAVE, compute the total number of passengers carried by the company's planes the past month, putting the answer in TOTAL. IR1 and IR2 are available for address modification and the completion test, respectively, in processing the arrivals. IR3 and IR4 are available for address modification and the completion test, respectively, for processing departures.

PART FIVE: THE LDX COMMAND



Now let's see how LDX works. Suppose we have the following five factors that make up one man's gross pay: regular pay, proficiency pay, overseas pay, hazardous duty pay, and clothing allowance. The amount he earns for each factor is stored in memory relative to the location PAY. Using address arithmetic, put the total in location GROSS.

To solve the problem, we will first load two index registers with 5 and 0, by using the LDX command. Because there are only 5 factors, we know that our loop counter will be 5. So, using the pair of index registers IR3 and IR4, our first instruction is:

<u>Op Code</u>	<u>First</u>	<u>Second</u>	<u>Third</u>
LDX	5	, IR3	, 0

The LDX command, like the TRX command, automatically pairs two successive index registers. Thus, the IR3 given in the second address is paired with IR4, the next highest IR.

When the program encounters the instruction:

LDX 5,IR3,0

the number in the third address is loaded into the index register specified in the second address (the address modifier), and the number in the first address is loaded into the paired index register (the loop counter). Thus, the 0 in the third address is loaded into IR3, and the 5 in the first address is loaded into IR4 (the paired index register).

Now that we have loaded the index registers, we must zero the accumulator.

What is the instruction for this? _____

Next, we add the first factor of the man's pay. Because this is the ADD instruction we will later modify in order to loop back, we will tag this instruction with IR3. So what does our program up to now look like?

Now we can use a TRX instruction to test for completion and to modify IR3 by 1:

TRX *-1,IR3,1

We finish our program by putting the sum in location GROSS and halting the computer:

STR GROSS
HLT

Now let's desk check our problem. Remember, we know that the 5 factors are stored in memory relative to location PAY. Let's assume that regular pay (PAY) equals \$200, proficiency pay (PAY+1) equals \$60, overseas pay (PAY+2) equals \$20, hazardous duty pay (PAY+3) equals \$55, and clothing allowance (PAY+4) equals \$5. We will use index registers IR1 and IR2.

So our program is:

```
LDX 5,IR1,0
CLA ZRO
ADD PAY,IR1
TRX *-1,IR1,1
STR GROSS
HLT
```

Now let's see if it works.

LDX 5,IR1,0 When the LDX instruction is carried out, IR1 and IR2 are automatically paired. IR1 is loaded with the 0 from the Third Address Field, and IR2--the paired IR--is loaded with the 5 from the First Address Field.

CLA ZRO The accumulator is zeroed to prepare for addition.

ADD PAY,IR1 The ADD instruction will add the contents of the address specified in the first field as augmented by IR1. Because IR1 contains only zero, the contents of PAY are added to the accumulator, giving the sum of \$200.

CLA ZRO

```
LDX 5,IR3,0
CLA ZRO
ADD PAY,IR3
```

TRX *-1,IR1,1 The TRX instruction pairs IR1 and IR2; then IR2 is tested for zero. IR2 contains a five on this first test, so it is decreased by one (for the first pass through the loop), then retested for zero. Since the contents of IR2 now equals four, the constant from the Third Address Field (a 1) is loaded into IR1 (IR1 now contains a 1) and the program transfers to the address specified in the first field, *-1 -- that is, the ADD PAY,IR1 instruction. The ADD instruction is carried out again. This time, because IR1 contains a 1, the effective address is PAY+1. So the accumulator now contains \$260, the sum of PAY and PAY+1. Again, the

TRX instruction is carried out. IR2 is tested for zero; since it contains a 4, it is decreased by 1 for the second pass and retested for zero. Now IR2 holds a 3, so IR1 is increased once more by the constant in the Third Address Field. Now IR1 contains a 2. The program transfers to *-1--the ADD PAY,IR1 instruction--and the contents of PAY+2 is added to the accumulator, giving the sum of \$280. For the third time the TRX instruction is carried out and the program loops back to the ADD PAY,IR1 instruction. The looping cycle continues until all 5 factors have been added to the accumulator, giving the gross pay of \$340. When the program has completed the ADD PAY+4 instruction, the TRX test for zero finds a zero in IR2 (which is the loop counter), so the program steps to the next instruction in the sequence.

STR GROSS The gross pay is stored in location GROSS (\$340).

HLT Stops the computer.

Just for practice, using the LDX command, write the instruction which would load IR3 with a 3 and IR2 with a 0. (IR2 is the address modifier and IR3 is used in the completion test.) The command would be:_____.

Now, back to the original program: We can make one more refinement to the program that uses the LDX command. Let's rewrite our program like this:

```
LDX 4,IR1,0
CLA PAY
ADD PAY+1,IR1
TRX *-1,IR1,1
STR GROSS
HLT
```


Notice that we put a 4 in the First Address Field of the LDX instruction instead of a 5. This means that we will loop through the program only four times. We can do this and still get gross pay, including all five factors, because we have changed the CLA ZRO instruction to CLA PAY.

So our program now loads IR1 and IR2 with 0 and 4, respectively; then it clears and adds PAY to the accumulator. Now we need only to add the other four factors of the man's gross pay.

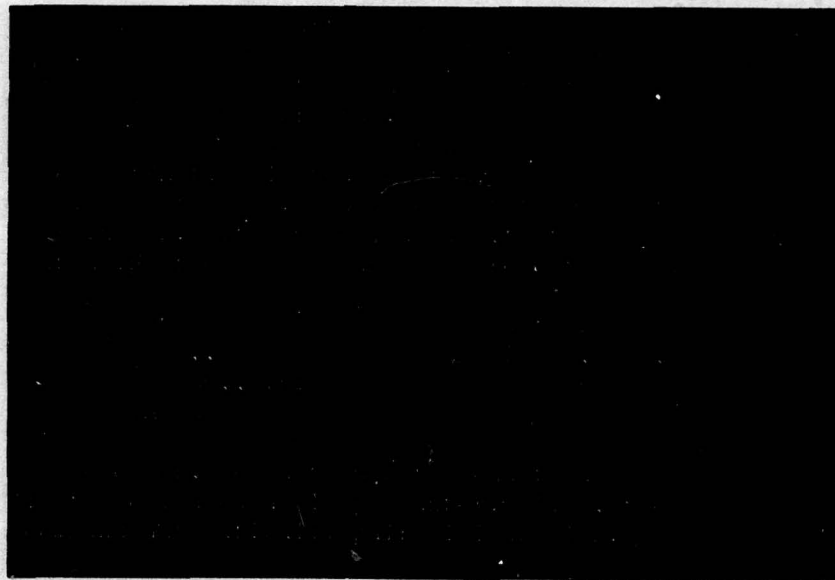
When we did not know the number to load in the index register that was to be the loop counter, we could not use this procedure, or the LDX command. When it is possible to use this procedure, it should be used because it eliminates one pass through the loop. In long complex problems the advantage in efficiency is desirable. But remember, you have to know the exact number of passes to be made before you can use the LDX command.

Problem 5.1. A large music store sells records of many different types, such as classical, western, folk, and so on. The number of each type is stored relative to TYPE; i.e., classical in TYPE, western in TYPE+1, etc. There is a 6 in TUNES, which is the number of different types of records they have. Using LDX, write a program to count the total number of records they have on hand at present. Put the answer in TOTAL. Use index registers IR1 and IR2 for address modification and the completion test, respectively.

LDX 3,IR2,0

Problem 5.2. A company keeps records on each man's gross pay stored relative to PAY. The factors which make up each man's gross pay are: regular pay, over-time pay, vacation pay, and his bonus. Write a program to compute the man's gross pay for the year, storing the answer in GROSS. Use LDX, and use IR2 for address modification and IR3 for the completion test.

Problem 5.3. Now do Problem 5.2 using the second method shown in this section. Remember, by this method you only have to loop through the program three times instead of four times.



PART SIX: REVIEW AND PRACTICE

We have now learned three different techniques for address modification. The first uses only a symbolic location name in the program:

```
REPEAT  CLA PAY
-----
-----
      TRU REPEAT
```

The second method uses symbolic location plus address arithmetic:

```
      CLA BOOK
      ADD PAGE
      STR BOOK
BIND   CLA TEMP
-----
-----
      STR BIND-3
```


Our third method uses the asterisk (*) plus address arithmetic:

```

TRZ *+5
CLA PAY
ADD ONE
STR PAY
TRU COMP
HLT

```

Each of these techniques causes the program to look back through a set of instructions, after the address field of an instruction word has been changed.

Problem 6.1. Rewrite the following program using symbolic location plus address arithmetic.

Using only symbolic location:

```

                                CLA STOCK
                                STR TEMP
                                CLA ZRO
                                STR VALSTK
COMPUT  CLA VALSTK
SUM      ADD VALUE
                                STR VALSTK
                                CLA TEMP
                                SUB ONE
                                STR TEMP
                                TRZ STOP
                                CLA SUM
                                ADD ONE
                                STR SUM
                                TRU COMPUT
                                STOP  HLT

```

Problem 6.2. Now then, rewrite Problem 6.1 using the asterisk plus address arithmetic.

Index registers are used as special temporary locations into which numbers can be copied to save the original contents of an address. IRs use two address fields, the first and the second, which must be separated by a comma. In effect, an instruction which uses an index register says to take the contents of the first address and carry out the operation STR, ADD, etc., on the contents of the second address.

Problem 6.3. Write an instruction that says the same thing as: ADD PAY+3. IR3 contains a 3.

Problem 6.4. Write an instruction that will say: STR COST+4, if IR2 contains a 4.

We used IRs for temporary storage of the loop counter and also for address modification.

The LOD command allows us to load an index register without using the accumulator and without two instructions:

```
CLA ZRO
STR IR1
```

The First and Third Address Fields are used with the LOD command and must be separated by two commas.

Problem 6.5. Write an instruction to copy the contents of VALUE into IR4.

The LOD command works like a STR command in that the number that is copied, the number in the location indicated by the first address, is not destroyed or changed. But anything in the index register specified by the third address is destroyed.

Problem 6.6. A supply depot maintains a 3-word record for each vacuum tube in stock. The words are: size, cost, and location--stored relative to TUBE. Using LOD, write a program to compute the total cost of tubes in stock. The number of tubes in stock is in TOTUBE. Use IR1 and IR2 respectively for address modification and the completion test, and store the answer in VALUE. Location KON1 contains a 1 and KON3 a 3. Use REPEAT as symbolic location for looping.

Problem 6.7. An electronics warehouse wants a program that will count the total number of vacuum tubes that they have ordered for the past month. Information concerning the vacuum tubes is stored relative to INFO. This information is: Part number, date ordered, shipment number, and number ordered. Use the LOD command and write a program to compute the total number ordered, storing the answer in TOTAL. The number of days in the month is in DAY. Use AGAIN as symbolic location for looping, IR1 for address modification, IR2 for the completion test. ONE contains a 1 and FOUR contains a 4.

The TRX command will modify two index registers at the same time, increasing one for address modification and decreasing the other for the completion test. The TRX command acts on the contents of an index register rather than the contents of the accumulator. This command uses all three address fields (First, Second, and Third):

TRX *-1,IR2,1

It also pairs, or links, successive IRs: IR1 and IR2, IR2 and IR3, etc.

Problem 6.8. A music store sells records of different types, such as classical, western, folk, and so on. The number of each is stored relative to TYPE; i.e., classical in TYPE, western in TYPE+1, and so on. The number of different types is in TUNE. Using TRX, write a program to compute the total number of records on hand at present. Put the answer in ANSWER; use IR2 for address modification and IR3 for the completion test.

Problem 6.9. An apartment building stores all the revenue for the month from its apartments relative to INCOME. Write a program, using the TRX command, to get the total income for the month. Use IR3 and IR4 respectively for address modification and the completion test. Their total number of apartments is in RENT. Store the answer in ALL.

The LDX command allows us to load two index registers at the same time; but it has one limitation: We must know the exact number of "loops," or "passes," needed for a particular problem. The LDX command will load the number of loops required for our computation.

Like the TRX command, LDX uses all three address fields and links successive IRs.

The LDX command loads two index registers at the same time.

LDX 5,IR1,0

This loads a 5 into the paired IR, or IR2, and loads a 0 into IR1.

Problem 6.10. A large business firm keeps the pay records for each employee. The salary of the first man is in PAY, the second man's pay is in PAY+1, etc.; i.e., the pay records are stored relative to PAY. There are 70 men employed by the company; this number is stored in MEN. Use the LDX command and write a program to compute the total pay of all employees, storing the total in TOTSAL. IR4 and IR1 are available respectively for address modification and the completion test.

Problem 6.11. There are 20 different uniform parts in stock. The cost of the first part is in PRICE, the second part in PRICE+1, and so on. TOTAL contains a 20. Using LDX, write a program to add the total cost of all the parts, storing the answer in ALL. IR1 and IR2 are available respectively for address modification and the completion test.

Problem 6.12. There are two kinds of vacuum tubes in an inventory, 6SN7 tubes and 6AQ6 tubes. Each tube has information on its type stored relative to TUBE. Each location in the series has either a 2 to indicate a 6SN7 tube or a 1 to indicate a 6AQ6 tube. The total number of all tubes is in STOCK. The cost of a 6SN7 tube is in VALUE and the cost of a 6AQ6 tube is in VALUE+1. Write a program, using the TRX command, to compute the total value of tubes in inventory, storing the total in TOT. Index registers One and Two are available respectively for address modification and the completion test. KON contains a 2. Use the TRN command.

Problem 6.13. Headquarters wants to know the number of second lieutenants eligible for promotion to First Lieutenant at the end of this month. Eighteen months active duty in grade are required for promotion from Second to First Lieutenant. The data for officers are stored in 5-word records as follows: rank, service number, months in grade on active duty, MOS, and assigned unit. These records are kept relative to RATING. Rank for officers is coded by a 1 for Second Lieutenant, 2 for First Lieutenant, and so on. PERSON contains the number of officers. TIME contains a 17. Store the number of eligible officers in UP. Use the TRX command. IR3 and IR4 are available respectively for address modification and the completion test. KON contains a 1.

Problem 6.14. There are two types of uniforms in an inventory, listed as big and small. Each uniform has information on its type stored relative to UNI. Each location in the series has either a 1 to indicate a big uniform or a 0 to indicate a small uniform. The total number of all uniforms in stock is 90. The cost of a big uniform is in VALUE, and the cost of a small uniform is in VALUE+1. Write a program, using the TRX and the LDX commands, to compute the total value of uniforms in inventory, storing the total amount in INV. Index Registers Two and Three are available respectively for address modification and the completion test.

Problem 6.15. A large music store sells records of many different types, such as classical, western, folk, and so on. The number of each type is stored relative to TYPE; i.e., classical in TYPE, western in TYPE+1, etc. There is a 6 in TUNES, which is the number of different types of records they have. The cost of each record is the same, the amount stored in COST. Write a program to compute the total number of records on hand, storing the answer in TOTAL, and also the total cost of the records, storing this in VALUE. Use IR1 and IR2 in computation of the total number, and IR3 and IR4 in computation of total cost. Use TRX and LDX whenever possible.